MATLAB 语言应用系列书

MATLAB 应用程序接口用户指南

刘 志 俭 等编著

科 学 出 版 社

2000

内 容 简 介

本书是《MATLAB 语言应用系列书》之一。全书共分八章，其中前七章在对 MATLAB 进行简要介绍的基础上，详细而系统地介绍了 MATLAB 应用程序接口的使用，内容包括如何在 MATLAB 环境下调用现有的用 C 语言和 FORTRAN 语言开发的算法；如何完成 C 语言和 FORTRAN 语言应用程序与 MATLAB 间的数据交换；以及如何在 C 语言和 FORTRAN 语言的应用程序中调用 MATLAB 中的各种函数，包括工具箱函数。同时为了方便用户对 MATLAB 应用程序接口的使用，书中对 MATLAB 应用程序接口中所提供的接口函数进行了详细说明。本书第八章在简要分析 MATLAB应用程序接口和 MATLAB C＋＋数学函数库之间异同的基础上，对 MATLAB C＋＋数学函数库进行了简单的介绍，内容包括类 mwArray 的说明、阵列对象的操作、库函数及运算符的使用和简单的程序设计。

本书可作为高等学校数学、计算机、电子工程、信息工程、机械工程等专业师生的参考教材，对从事上述领域工作的广大科技工作者和开发应用人员具有重要的参考应用价值。

图书在版编目(CIP)数据

MATLAB 应用程序接口用户指南/刘志俭等编著.-北京：科学出版社，2000

MATLAB 语言应用系列书

ISBN 7-03-008669-4

Ⅰ.M… Ⅱ. 刘… Ⅲ. 程序语言，MATLAB Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2000）第 65346 号

科学出版社 出版

北京东黄城根北街 16 号

邮政编码：100717

北京双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2000 年 8 月第 一 版 开本：787×1092 1/16

2000 年 8 月第一次印刷 印张：26 1/2

印数：1—4 000 字数：613 000

定价：35.00 元

前　　言

自 1984 年 MathWorks 公司首次推出 MATLAB V1.0 版本，到目前为止的 MATLAB V5.3 版本，MATLAB 采用了开放性开发的思想，不断吸收各学科领域权威人士所编写的实用程序，经过不断地发展和扩充，MATLAB 现已发展成为国际上最优秀的科技应用软件之一。其强大的科学计算与可视化功能、简单易用的开放式可扩展环境以及多达 30 多个面向不同领域而扩展的工具箱（Toolbox）支持，包括了通信系统、信号处理、图像处理、小波分析、鲁棒控制、系统辨识、非线性控制、模糊控制、神经网络、优化理论、样条、商用统计分析等等大量现代工程技术学科的内容，使得 MATLAB 在许多学科领域中成为计算机辅助设计与分析、算法研究和应用开发的基本工具和首选平台。在我国，MATLAB 已经拥有许多用户，许多高校陆续开设了有关 MATLAB 的课程，清华大学、华中理工大学等高校的 BBS 上还专门设立了 MATLAB 讨论区。

MATLAB 应用程序接口（MATLAB Application Program Interface）是 MATLAB 系统提供的一个非常重要的组件，通过该接口，用户可以方便地完成 MATLAB 与外部环境的交互。MATLAB 应用程序接口主要包括三部分内容，分别为 MEX 文件——外部程序调用接口，MAT 文件应用程序——数据输入输出接口和 MATLAB 计算引擎函数库，它们实现的功能分别为

· 在 MATLAB 环境中调用 C 语言或 FORTRAN 语言编写的程序，以提高数据处理的效率；

· 向 MATLAB 环境传送数据或从 MATLAB 环境接收数据，即实现 MATLAB 系统与外部环境的数据交换；

· 在 MATLAB 和其他应用程序间建立客户机/服务器关系，将 MATLAB 作为一个计算引擎，在其他应用程序中调用，从而降低程序设计的工作量。

MATLAB C/C＋＋数学函数库是 MATLAB 扩展中的重要组成部分，其中包含了约 400 个 MATLAB 数学函数，用户按照一定的规则，可以在 C 语言和 C＋＋语言的应用程序中轻松地调用它们，用于完成原本非常繁杂的矩阵运算任务，从而极大地减少程序设计的工作量。更为重要的是，使用 MATLAB C/C＋＋数学函数库编写的应用程序可以完全脱离 MATLAB 环境独立地执行，与 MATLAB 应用程序接口相比，这是其最大的优点。

本书的具体组织如下：

第 1 章对 MATLAB 进行简要的介绍；

第 2 章至第 7 章详细地讲述了 MATLAB 应用程序接口的使用，并对所有的库函数进行了说明；

第 8 章对 MATLAB C＋＋数学函数的基本概念及其使用进行了一般的介绍。

本书中第 5 章的第 1、第 2 节由张志勇编写，其余各章均由刘志俭编写。

在本书的编写过程中，得到了科学出版社鞠丽娜编辑的大力支持，并为本书的校对作了大量的工作，在此向她表示衷心的感谢。同时也要感谢父母和老师多年来对我的培养和教育，感谢女友刘毅小姐对我的关心和支持，感谢国防科技大学、华中理工大学、上海交通大学和清华大学各位钻研 MATLAB 的网友们给我的启发。

作者学识有限，希望读者对书中的错误不吝赐教。

刘志俭

2000 年 4 月

目　　录

第 1 章　MATLAB 系统及程序设计简介

本章将从四个方面对 MATLAB 系统进行较为全面的介绍：首先介绍 MATLAB 系统的产生、MATLAB 系统的构成以及 MATLAB 的共生产品，让读者对 MATLAB 系统有一个整体的了解；然后介绍 MATLAB 系统所使用的数据结构；再次简单地介绍 MATLAB语言程序设计；最后介绍基本的 MATLAB 矩阵操作。熟悉 MATLAB 系统的读者可以越过本章内容，从第 2 章开始阅读本书。

1.1　MATLAB 系统简介

1.1.1　MATLAB 系统的产生

使用过C、C++和 FORTRAN 等高级程序语言进行算法开发的读者可能已经知道，当算法中涉及到对矩阵的处理、运算和一些绘图操作时，程序设计将是一件非常麻烦的任务，这主要是因为两点：第一，这些高级程序设计语言本身并不包含矩阵类型的数据结构(注意，这里所说的数据结构仅仅是指那些程序设计语言本身所包含的数据结构，而不包含第三方开发的一些数据类型，如 C++矩阵类等)，这样，当读者希望进行一些关于矩阵的算法设计时，首先需要完成的任务就是将矩阵的元素一一读人，并以它们为处理和计算的对象，而并非以矩阵整体为对象，这个步骤将由一系列的循环过程来完成，非常麻烦；第二，在这些高级程序设计语言中，没有嵌入自身的标准计算子程序库和函数库，这样读者在进行算法设计时，往往需要重复地编写一些代码来完成固定的功能，这是一个非常费时和枯燥的工作。

MATLAB 的首创者 Cleve Moler 博士在新墨西哥大学讲授线性代数时，发现了同样的问题，便萌生了开发 MATLAB(MATrix LABoratory)——矩阵实验室的想法，并将之付诸实施，利用 FORTRAN 语言和当时极为流行的基于特征值计算的软件包 EISPACK 以及线性代数软件包 LINPACK 中大量可靠的子程序，编写了这一集命令翻译、科学计算于一身的交互式软件。在 MATLAB 下，繁琐的矩阵处理和运算变得异常容易。

随着 MATLAB 的应用范围越来越广泛，Moler 博士等数学家与一些软件专家成立了现在的 MathWorks 公司，致力于开发、扩展和改进 MATLAB 系统。而后由于采用了开放式的开发思想，不断吸收各学科领域权威人士所编写的实用程序，如今的 MATLAB 不但已经完全用 C 语言进行了全面的改写，增添了丰富的图形图像处理和多媒体功能，而且形成了一个规模庞大、覆盖面极广的工具箱(Toolbox)，其内容包括了信号处理(Signal Processing)、图像处理(Image Processing)、小波分析(Wavelet)、鲁棒控制(Robust Control)、系统辨识(System Identification)、非线性控制(Non-linear Control)、模糊逻辑(Fuzzy Logic)、神经网络(Neural Network)、优化理论(Optimization)、样条(Spline)、μ 分析和综合(μ-Analysis and Synthesis)、统计分析(Statistics)等等大量现代工程技术学科的内容，极为实用。1992 年，MathWorks 公司推出了交互式模型输入与仿真环境

Simulink，为 MATLAB 在仿真和 CAD 中的应用开创了新的局面。该公司于同年推出了具有划时代意义的 MATLAB4.0 版本，并于 1993 年推出了其基于 PC 机平台的 Windows 版本，并且不断地升级改进，现在已经推出了 5.3 版本，不但适用于各种硬件平台和操作系统，而且功能得到了进一步的增强，成为一个高度集成的，集科学计算、程序设计和可视化于一身的软件环境。在该软件环境中，问题和问题的解答均以人们熟悉的数学形式表示出来，极为易用，其典型应用包括：

- 数学和计算；
- 算法开发；
- 建模和仿真；
- 数值分析，检测和可视化；
- 应用程序开发(包括图形用户界面)。

此外，MATLAB 在控制界、生物医学工程、语音处理、图像信号处理、雷达工程、信号分析、计算机技术等各行各业中也得到了广泛的应用。

1.1.2 MATLAB 系统的构成

MATLAB 系统主要由五大部分组成，分别为 MATLAB 语言(the MATLAB Language)，MATLAB 工作环境(the MATLAB Working Environment)，MATLAB 数学函数库(the MATLAB Math Library)，图形句柄系统(Handle Graphics®)和 MATLAB 应用程序接口(the MATLAB Application Interface)。下面对它们进行分别介绍。

1. MATLAB 语言

MATLAB 语言是一种以矩阵(matrix)和阵列(array)为基本编程单元的，拥有完整的控制语句、数据结构、函数编写与调用格式和输入输出功能的具有面向对象程序设计特征的高级程序语言。读者不但可以利用它方便快捷地完成小规模的算法验证、程序开发和调试工作，而且可以使用它进行大规模的复杂应用程序设计，非常有效。

与其他高级程序设计语言相比，MATLAB 语言除了执行效率要低于高级语言之外，无论是在编程的效率、可读性还是可移植性方面都要远远高于其他高级语言，对于科技工作者来说，是一种非常实用的编程工具。而且由于 MATLAB 语言轻松地实现了 C 语言和 FORTRAN 语言的几乎全部功能，并且在图形功能方面有所加强，同时提供了大量的功能齐全的数学函数，不但可以使用户编制出功能强大、界面优美的应用程序，而且可以极大地缩短开发周期。但是严格说来，MATLAB 语言并不是一种真正的计算机语言，因为用它所开发的程序不能脱离 MATLAB 的解释性执行环境而运行。

相关的 MATLAB 语言的数据类型和 MATLAB 语言的语法将分别在 1.2 节和 1.3 节中进行介绍。

2. MATLAB 工作环境

MATLAB 工作环境简而言之就是一系列实用工具的集合，它不但包括了各种操作工作空间中变量的工具和管理数据输入输出的方法，而且包括了开发调试 M 文件和 MATLAB 应用程序的集成环境，使用起来极为方便。当用户在 Window NT 系统下启动

MATLAB后，将会出现如图1.1所示的命令窗口(the Command Window)，这是用户同MATLAB工作环境交互的主要窗口，在命令提示符()下，用户可以键入各种相关命令，来完成所希望的操作。例如为了求得矩阵$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$的转置矩阵，可在提示符下键入如下两条命令：

```
? A=[1,2;3,4];A=A'.
```

第一条命令终止于中括号后的第一个分号，用于构造矩阵A，其中中括号为矩阵的构造算符，逗号用于将同一行中的元素分开，而分号则用于区别不同的行；第二条命令用于求矩阵A的转置，其中为转置运算符。当键入回车(Enter)后，MATLAB将显示如下结果：

```
A=
    1  3
    2  4
```

这里注意一点，在MATLAB中，同一命令行中可以键入多条命令，中间用逗号或分号隔开，它们的区别是：用逗号时，MATLAB将回显每一条命令的执行结果，而用分号则不会回显。有关MATLAB的具体应用，本书不再赘述，读者可以参阅相关的参考书籍。

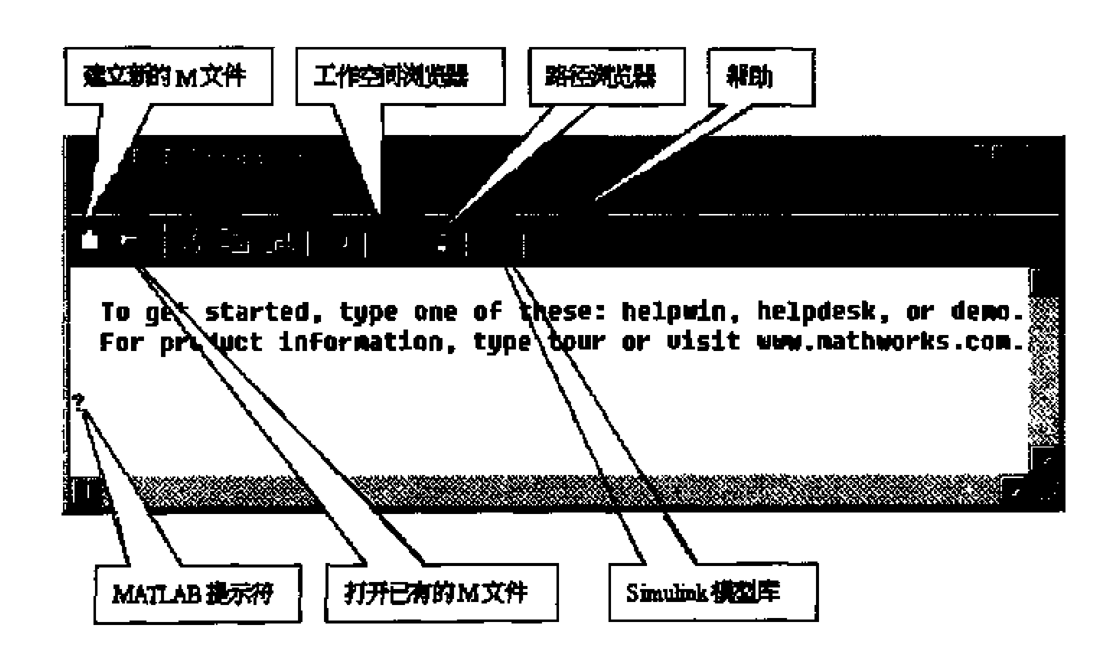


图1.1 MATLAB命令窗口

在命令窗口中，用户除了可以在命令提示符下键入命令执行操作外，还可以通过菜单和工具条执行多种任务，如图1.1所示，通过建立新的M文件按钮和打开已有的M文件按钮可以开启Medit编辑调试器，如图1.2所示，这是一个功能非常完善的编辑调试环境；通过工作空间浏览器按钮，可以开启工作空间浏览器，察看当前工作空间浏览器中各变量的类型和内容；通过路径浏览器按钮，可以开启路径浏览器，让用户设置当前MATLAB工作环境所使用的默认路径；通过帮助按钮，可以打开帮助窗口，让用户查找在线帮助；通过Simulink模型库按钮，可以打开Simulink模型库，让用户向自己的模型中添加新

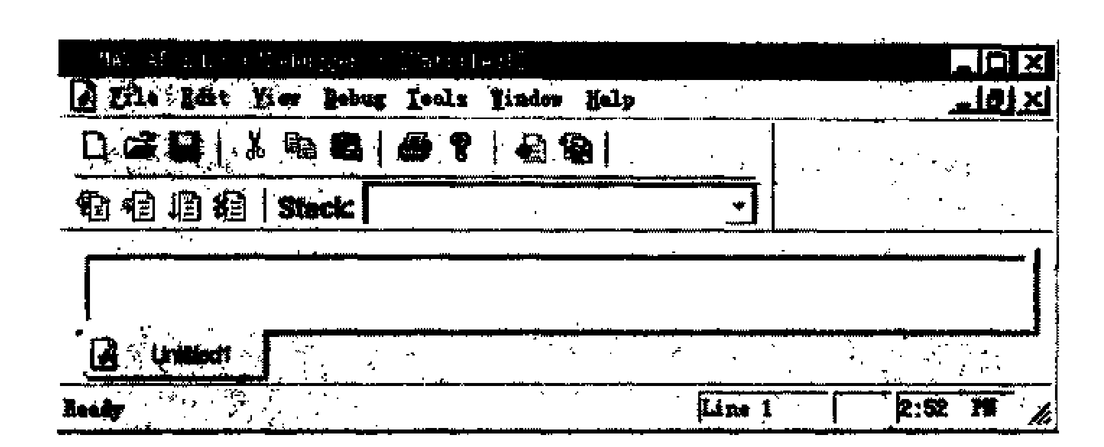


图 1.2 Medit 编辑调试窗口

的模块。总之，MATLAB 工作环境是一个功能异常强大的工具集合，可以令用户完成几乎所有的操作，并且简单易用。

3. MATLAB 数学函数库

MATLAB 数学函数库是大量的各种形式的数学函数和算法的集合，它不但包括了最基本的初等函数，如 sum、sine、cosine 和复数运算等，而且包含了大量复杂的高级函数和算法，如贝塞尔(Bessel)函数，快速傅里叶变换和矩阵求逆等。用户在编写自己的 MATLAB 程序时，可以轻松地调用这些函数和算法，从而极大地方便了算法的开发。所有这些函数按类别分别存放在 MATLAB 工具箱目录下的八个子目录中，详见表 1.1。

表 1.1 MATLAB 数学函数库的分类和组织

目 录 名	函 数 功 能
elmat	对矩阵和矩阵元素的操作
elfun	初等数学函数
specfun	专门数学函数
matfun	矩阵函数——数值线性代数
datafun	数值分析和傅里叶变换
polyfun	插值和多边形近似
funfun	功能函数和 ODE 求解
sparfun	稀疏矩阵函数

4. 图形句柄系统

Handle Graphics®，为 MathWorks 公司的注册商标，是 MATLAB 的图形系统。它在包含了大量高级的 2D 和 3D 数据可视化、图形显示、动画生成和图像处理命令的同时，还提供了许多低级的图形命令，允许用户按照自己的需求显示图形和定制应用程序图形用户接口，既不失方便，又不失灵活。具体的函数分为五大类，分别放置于 MATLAB 工具箱

下五个不同的目录内，详见表 1.2。

表 1.2　MATLAB 图形函数的分类和组织

目录名	函数功能
graph2d	二维图形函数
graph3d	三维图形函数
specgraph	专用图形函数
graphics	图形句柄函数
uitools	图形用户界面工具

5. MATLAB 应用程序接口

MATLAB 应用程序接口是 MATLAB 为用户提供的一个功能完善的函数库，它包含了大量的 MATLAB 与 C 语言和 FORTRAN 语言之间的接口函数，是 MATLAB 的一个非常重要的组成部分。通过它，不仅可以在 MATLAB 下以动态链接库的形式调用 C 语言或 FORTRAN 语言编写的子程序，而且可以在 C 语言和 FORTRAN 语言中调用 MATLAB 的大量函数，将 MATLAB 作为一个计算引擎，同时还能够完成 MATLAB 与外界必要的数据交换，极大地增强了 MATLAB 的灵活性，非常实用。本书将着重讲述这一部分内容，使读者能够熟练地应用 MATLAB 应用程序接口。

1.1.3　MATLAB 共生产品

由图 1.3 所示的 MATLAB 产品家族可以看到，MATLAB 产品家族是一个非常庞大的系统，MATLAB 系统仅仅是其中的一个部分，它还有许多其他重要的成员，如 Simulink 等，下面我们对它们进行一些简单的介绍。

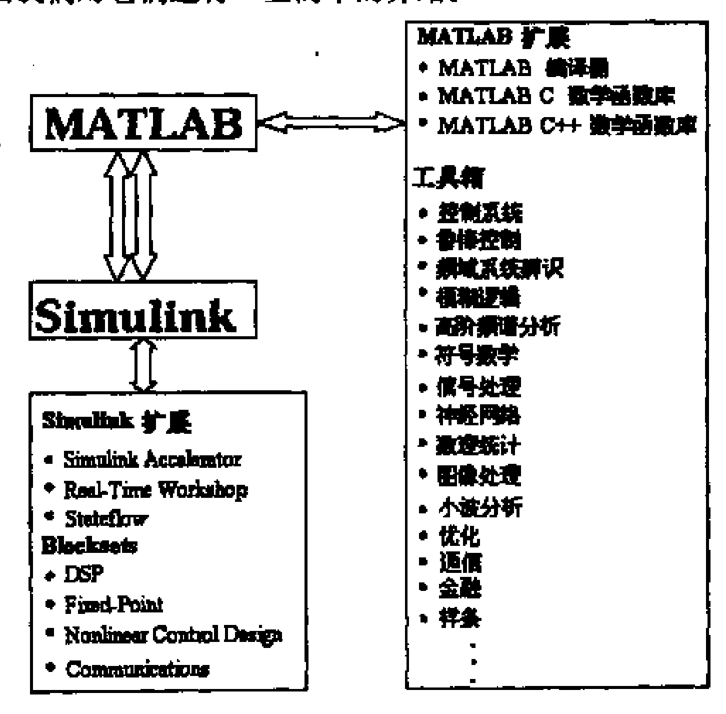


图 1.3　MATLAB 产品家族

1. Simulink 及其扩展

Simulink 是一个用来对非线性动态系统进行仿真的、鼠标驱动的交互式图形系统，它允许用户通过绘制一系列的方框图来完成建模工作，并动态地对模型进行操作，适用于各种系统，包括线性系统、非线性系统、连续系统、离散系统和多变量系统，是 MATLAB 系统的一个非常重要的共生产品。

Blocksets 是 Simulink 的一个插件集，它为 Simulink 提供了大量额外的专用模块库，如信号处理、通信等。

Real-Time Workshop 是一个非常实用的应用程序，它可以由用户的方框图生成 C 语言的代码，并且能够运行在各种各样的实时系统上。

2. MATLAB 编译器

MATLAB 编译器是 MATLAB 系统扩展的重要组成部分，通过它，用户可以将 MATLAB 的 M 文件转化为 C 或 C++语言的源代码，增强了 MATLAB 应用的灵活性。转换后的源代码主要有以下三种类型：

- 可生产 MEX 文件的 C 语言源代码；
- 可与其他模块结合，生成可执行程序的 C 或 C++源代码，所生成的可执行程序可以独立于 MATLAB 的解释性环境单独运行，但是需要 MATLAB C/C++数学函数库的支持；
- 用于 Simulink 和 Real-Time Workshop 的 C 语言 S 函数。

3. MATLAB C/C++数学函数库

MATLAB C 数学函数库是 MATLAB 系统扩展的另一重要组成部分，包含了大约 400 个用 C 语言进行重新编写的 MATLAB 数学函数，不但包括大量的 MATLAB 内建函数，而且包含了许多 MATLAB 的 M 文件，是 MATLAB 系统数学计算核心的 C 语言的再现。任何能够调用 C 语言函数的程序，均能够调用该函数库所包含的所有数学函数，为应用程序开发者提供了一种方便的使用 MATLAB 强大计算能力的途径，其核心结构是 mxArray 结构体。这里需要明确的一点是该函数库包含的仅仅是数学函数，并没有包含其他的一些专用函数，如图形句柄系统函数、Simulink 函数等。

MATLAB C++数学函数库的功能与 MATLAB C 数学函数库的功能相同，不过它是构建在 MATLA C 数学函数库的上层，用 mxArray 类代替了 mxArray 结构体，对许多功能进行了封装。在本书的最后一章，我们将对 MATLAB C++数学函数库的使用进行简要的说明。

4. MATLAB 工具箱

MATLAB 工具箱是由一系列各式各样的函数库组成，内容涉及方方面面，包括了大量的 M 文件和 MEX 文件，主要由各行各业的专业人士编写，其目的是为了方便某一领域内的科学研究和工程应用，将一些已经非常成熟或完善的算法标准化供人调用。到目前为止，由 MathWorks 公司正式发布的工具箱已达 22 个，图 1.3 中仅仅列出了其中的一部

分，可见其涉及的领域是相当广泛。而且由于采用了开放式的开发思想，在因特网上，有相当多的个人或团体将自己开发的工具箱置于网上，供人借鉴和免费下载，从而形成了巨大的资源宝库，同时也促进了 MATLAB 的发展。

1.2 MATLAB 的数据类型

在上一节中，我们对 MATLAB 系统作了全面的介绍，讲述了 MATLAB 系统的产生、MATLAB 系统的组成以及 MATLAB 系统的一些共生产品，使读者对 MATLAB 系统有了一个大概的了解。在本节中，我们将对 MATLAB 所使用的数据结构进行介绍，以加深读者对 MATLAB 系统的理解。在本节中，为了讲解方便，举例一般为一维或二维，但是同样可以推广到多维情况下。

1.2.1 MATLAB 阵列

MATLAB 阵列(array)是 MATLAB 语言惟一能够处理的对象类型，MATLAB 所有的基本数据类型，包括数量(scalar)，向量(vector)，矩阵(matrix)，字符串(string)，单元阵列(cell array)，结构体(structure)，都属于阵列对象，只不过是阵列对象的不同构成方式。在本小节中，我们将首先对 MATLAB 的数据存储方式进行说明，然后讲述一些简单的 MATLAB 基本数据类型。

1. MATLAB 数据的存储

所有的 MATLAB 数据类型的存储方式均为按列存储，这主要是因为最初的 MATLAB 系统采用的是 FORTRAN 语言进行编写，而 FORTRAN 语言的数据存储方式为按列存储。这与 C 语言按行进行数据存储的方式有着较大的差异，对于习惯使用 C 语言的读者来说，需要特别注意这一点。例如在 MATLAB 命令提示符下键入如下给定矩阵：

```
A = ['MATLAB' ; 'ABACUS' ; 'DEDUCE']
```

回车后得到如下结果：

```
A =
    MATLAB
    ABACUS
    DEDUCE
```

为一个 3×6 的矩阵。在 MATLAB 的内存中，它是按如下方式进行存储的：

```
M A D A B E T A D L C U A U C B S E
```

2. 数量、向量和矩阵

复数矩阵(complex matrix)是 MATLAB 中最常用和最基本的数据类型，大小可以表示为 m×n，其中 m 和 n 分别为矩阵的行数和列数。数量(scalar)和向量(vector)是特殊的矩阵形式，其中数量为 1×1 的单元素矩阵，向量则是维数为 1×n 或 m×1 的单行或单列的矩阵。按照矩阵元素的类型，可以将 MATLAB 矩阵分为以下几类：双精度浮点型，单

精度浮点型，逻辑型，8 位、16 位或 32 位有符号和无符号整数型，其中以双精度浮点型最为常用。在 MATLAB 中，双精度的复数矩阵用两个向量来表示，分别用来存储矩阵的实数部分和虚数部分，通过指针 pr（实部）和 pi（虚部）可以得到相应的数据。对于实数型的矩阵来说，存储和表示方法与复数型矩阵相同，只不过虚部指针为空（NULL）。同时，在 MATLAB 中还提供了一些可判断和转换矩阵数据类型的函数，如 isa 函数可以判断矩阵所属的数据类型，uint8 函数可以将任何类型的数值矩阵转换为用 8 位整数表示的矩阵，而 double 函数可以将矩阵转换为双精度浮点型。具体的函数使用格式，读者可参阅联机帮助。

3. 字符串

MATLAB 字符串（string）是 MATLAB 矩阵的又一种表现形式，其元素类型为用 16 位无符号整数表示的 Unicode ASCII 字符（character）。与 C 语言不同的是，在 MATLAB 中字符串不以空子符 NULL 来表示字符串的结束。如果 MATLAB 工作空间中存在一个 A = 'CDEF' 的字符串，使用 whos 命令，MATLAB 将显示如下结果：

```
Name     Size     Bytes     Class
A        1×4      8         char array
```

可以清楚地看到，MATLAB 系统将该字符串作为一个 1×4 的矩阵，并且占用 8 个字节的存储空间。

同时在 MATLAB 系统中，提供了相当多的对字符串进行操作和转换的内部函数（见表 1.3），使得对字符串的操作极为方便。

表 1.3 字符串操作及转换函数（部分）

函 数 名	功 能
char	构造字符串
blank	构造空格字符串
Ischar	判断矩阵是否为字符串类型
Strcmp	字符串比较
Strcat	字符串拼接
Upper	将字符串转换为大写
Lower	将字符串转换为小写
num2str	将数字转换为字符串

4. 稀疏矩阵

MATLAB 中，为矩阵提供了两种完全不同的存储方式，即满（full）和稀疏（sparse），在一般情况下，MATLAB 所创建的矩阵均为 full 的存储类型。

稀疏矩阵（sparse matrix）是 MATLAB 矩阵的一种特殊类型，它包含了大量的值为零的矩阵元素。正是由于这个特性，MATLAB 系统为稀疏矩阵提供了完全不同的存储方

式和处理方法，即只对那些非零的元素进行存储和计算，从而在很大程度上降低了对内存的需求量和处理的计算量。

考虑一个 m×n 的拥有 nnz 个非零元素的实稀疏矩阵，MATLAB 系统将使用三个内部阵列来表示该稀疏矩阵，其中第一个阵列按单精度浮点型来存储稀疏矩阵中所有的非零元素，其长度为 nnz；第二个阵列用来存储所有非零元素的行索引值，长度同样为 nnz，元素类型为整型；第三个阵列则是一个用来标示每一列开始的整型指针，其长度为 n。如果存储一个单精度型浮点数需要 8 个子节，而整型需要 4 个字节，这样对于一个 m×n 稀疏矩阵来说，其内存需求量为

8×nnz+4×(nnz+n)。

对于复数型的稀疏矩阵，则需要第四个阵列来存储非零矩阵元素的虚数部分，长度为 nnz。这里需要提醒读者注意一点，MATLAB 系统并不会自行创建稀疏矩阵类型，而需要用户通过命令行来显式建立。例如存在矩阵 $A=\begin{pmatrix}1&0&0&4\\0&2&0&0\\0&0&3&0\\0&0&0&5\end{pmatrix}$，读者可以在 MATLAB 命令提示符下键入命令 S = sparse(A)来将矩阵 A 转换为稀疏矩阵，回车后，可以得到如下结果：

```
S =
    (1,1)    1
    (2,2)    2
    (3,3)    3
    (1,4)    4
    (4,4)    5
```

反之，同样可以将一个稀疏矩阵显式转换为一个满矩阵，通过使用命令

```
A = full(S)
```

在实际的使用中，是否将一个矩阵转换为稀疏矩阵，应视非零元素的多少而定。如果将非零元素较多的矩阵转换为稀疏矩阵，不但达不到减少内存消耗的目的，反而会增加内存的使用量。所以在转换前，必须小心衡量。

5. 多维阵列

MATLAB 多维阵列(Multidimensional Array)是常规的 MATLAB 矩阵的一种扩展形式，其维数一般大于 2，除了使用常规的行下标和列下标外，对于多维阵列的元素的访问还需要使用更多的下标描述，通常将 3 或大于 3 的维描述为“页面”，其示意图见图 1.4，命令

```
A(2, 3, 1) = 5
```

的功能是将多维阵列中第一个页面、第二行、第三列的元素赋值为 5。这里必须特别注意一点，MATLAB 中对于下标的使用方式与 C 语言中完全不同，在 C 语言中，排在最前面的下标等级最高，依次向后递减；而在 MATLAB 中，排在第一第二的下标分别代表矩阵

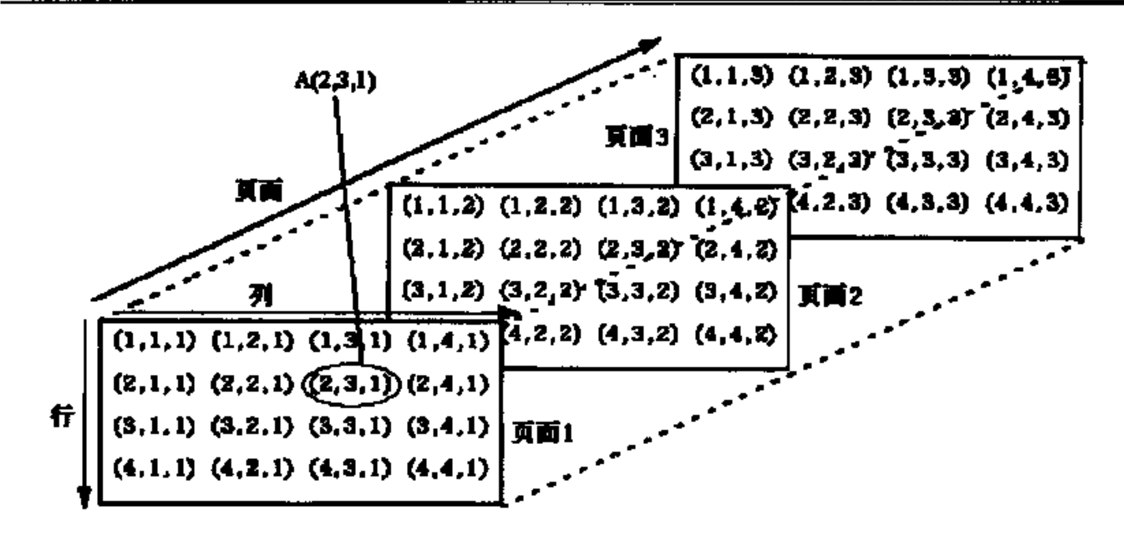


图 1.4 多维阵列示意图

的行和列，从第三个下标开始才为页面，而且排在越前面的下标等级越低，最后一个下标为最高级。

MATLAB 中一共提供了三种创建多维阵列的方法，其中第一种方法为直接通过下标索引来创建多维阵列，例如在 MATLAB 命令提示符下键入以下命令

```
A(:,:,2) = [1 0 4; 3 5 6; 9 8 7]
```

就相当于创建了一个三维阵列，并将其中第二个页面内的元素全部赋值，同时 MATLAB 通过使用算符“:”可以将一个页面内的所有元素赋为同一个值，命令如下：

```
A(:,:,2) = 5
```

第二种方法为使用 MATLAB 提供的阵列构造函数，例如 randn，repmat，ones 以及 zeros 等，命令

```
B = randn(4,3,2)
```

构造了一个数值为正态分布的具有两个页面的 4×3 的 3 维阵列，而命令

```
B = repmat(5,[3 4 2])
```

则创建了一个具有两个页面的 3×4 的 3 维阵列，并且将所有的元素赋值为 5；第三种方法为使用 cat 函数构造多维阵列，通过 cat 函数可以在指定维数的情况下，将多个阵列连结在一起从而构成多维阵列，例如命令

```
B = cat(3,[2 8; 0 5],[1 3; 7 9])
```

创建了三维阵列，其中第一个页面包含了矩阵[2 8; 0 5]，第二个页面包含了矩阵[1 3; 7 9]。

1.2.2 复杂的 MATLAB 基本数据类型

除了上一小节中的几种较为简单的 MATLAB 基本数据类型外，还有三种较为复杂的基本数据类型，它们分别是单元阵列、结构体和对象，在本小节中，我们将对它们分别加以介绍。

1. 单元阵列(cell array)

单元阵列是MATLAB系统中非常特殊的一种阵列类型,阵列的每一个元素称为一个单元(cell),每一个单元可以为各种类型的MATLAB基本数据类型,不但可以为上一小节中讲述过的简单的数据类型,而且可以为本小节中即将讲述的结构体以及对象,同时可以为单元阵列;更为重要的是单元阵列的各个单元之间可以互不相同,极为灵活。创建单元矩阵总的说来有两种方式:一种方式为直接对单元阵列的单元进行赋值,另一种为首先使用cell函数为单元阵列分配空间,然后进行赋值。例如在MATLAB命令提示符下,键入如下命令:

```
A(1,1) = {[1 4 3; 0 5 8; 7 2 9]};
A(1,2) = {'It is a cell'};
A(2,1) = {[1,2;3,4]};
A(2,2) = {-1;1};
```

可以得到如图1.5所示的单元阵列。

cell(1,1) 1 4 3 0 5 8 7 2 9	cell(2,1) It is a cell
cell(1,2) 1 2 3 4	cell(2,2) [-1,0,1]

图1.5 单元矩阵

2. 结构体

MATLAB结构体(structure)的定义与C语言结构体的定义类似,是由一系列不同的域(field)组成,每一个域均可以为不同的MATLAB基本数据类型,图1.6是一个典型的MATLAB结构体示意图。

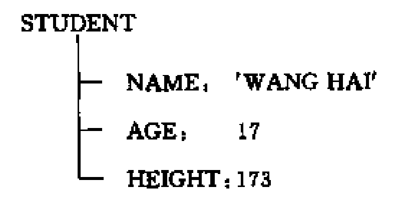


图1.6 MATLAB结构体

与其他数据类型相同,MATLAB系统同样将结构体视为一个阵列,其维数为1×1。构造结构体的方法不外乎两种,一种为直接赋值,另一种为使用struct函数。例如在

MATLAB 命令提示符下键入如下命令：

```
? STUDENT.NAME = 'WANG HAI';
? STUDENT.AGE = 17;
? STUDENT.HEIGHT = 173;
? STUDENT
```

回车后 MATLAB 系统将回显如下结果：

```
NAME: 'WANG HAI';
AGE: 17;
HEIGHT: 173;
```

这样就构造了一个名为 STUDENT 的结构体。使用 struct 函数的格式如下：

```
STRUCT_ARRAY = struct('field1', 'value1', 'field2', 'value2',…)
```

其中 field 代表域名，value 代表域值。访问结构体的内容可以使用如下格式：

```
field_value = STRUCT_ARRAY.field_name
```

读者同样可以构造结构体阵列，对其访问只需在结构体阵列名后加上索引即可，与 C 语言的结构体数组的定义和使用非常类似。

1.2.3　类(class)和类对象(object)

在 MATLAB 中，类(class)的定义与 C++语言中类的定义极为相似，是一组具有相似特征对象(object)的抽象，每一个对象都是类的一个实例。前面小节中讲述的各种 MATLAB 的基本数据类型都是 MATLAB 的内建类(见表 1.4)，用它们定义的变量称为该类的类对象。

表 1.4　MATLAB 的内建类

类　名	描　述
double	双精度浮点类型矩阵或阵列
sparse	二维稀疏矩阵
char	字符阵列
struct	结构体阵列
cell	单元阵列

通过类，用户可以构造新型的 MATLAB 数据类型，并且声明一系列对该类类对象的运算符和操作函数。正是因为有了类，才使 MATLAB 语言具有了面向对象编程语言的特征，因此对类和类对象概念的理解，对学习和使用 MATLAB 来说至关重要。在各种各样的 MATLAB 工具箱中，类更是得到了广泛的应用，例如 MATLAB 的符号计算工具箱就是建立在自定义类 sym 的基础上，该类定义了各种对符号矩阵和变量的操作；而在控制系统工具箱中，则是通过自定义的类 lti 以及它的三个子类来完成对线性时变系统分析的

任务。

1. 类的组成

一般来说，一个类可以分为两个组成部分，即成员变量和成员函数。成员变量相当于 MATLAB 结构体的域，可以为各种类型的 MATLAB 数据；而成员函数是一系列的 M 文件，可以包括各种类型的 MATLAB 运算符、数学算子、功能函数以及重载后的 MATLAB 内建算子和函数，用来对成员变量进行操作。类对象的存储结构与 MATLAB 结构体的存储方式完全相同，但是在对域内容的访问和保护上却截然不同，类对象的域内容只能通过类的成员函数来获取和修改，外界根本无法得知，这也就是面向对象程序设计的一个重要特性——数据的封装性。此外，在 MATLAB 系统下，类成员函数的存放有着相当严格的规定，所有的类成员函数必须放在同一目录下，且目录名必须为@class _ name（不同的操作系统格式可能存在差别，本书主要基于 Windows 系统），其中符号@为固定格式，不可省略，class _ name 为用户所定义类的类名；同时这个新建立的目录@class _ name 必须设置为 MATLAB 系统搜索路径中工作目录的子目录。

2. 类的构造函数

由于在 MATLAB 语言中，没有数据类型的声明语句，数据的类型一般在定义时隐式地通过构造函数来确定，并设置类型标志。例如如下定义

```
A = zeros ( 5 , 5 )
```

将矩阵 A 默认地定义为双精度浮点类型的 MATLAB 矩阵，而定义

```
STR = 'She is beautiful'
```

则将变量 STR 定义为 char 类的一个实例。因此在构造 MATLAB 新类时，类构造函数的建立非常重要。MATLAB 规定，类构造函数名必须与类名相同，并且存放在目录@class _ name 下。下面是一个简单的类构造函数 DEMO _ CLASS()，类 DEMO _ CLASS 仅包含一个类型为 double 的矩阵的域 A：

```
function p = DEMO _ CLASS (a)
    %DEMO _ CLASS class constructor.
    %p = DEMO _ CLASS (v) creates a matrix of v×v.
    if nargin == 0
        p.A = zeros(5,5);
        p = class(p, 'DEMO _ CLASS'); % Set class tag
    else
        if isa(a, 'DEMO _ CLASS')
            p = a;
        else
            p.A = zeros(a, a).';
            p = class(p, 'DEMO _ CLASS');
        end
    end
```

该构造函数在没用输入参数的情况下产生一个大小为 5×5 的 double 类型零矩阵，赋给 DEMO_CLASS 类对象 p，并且设置类型标志。

本小节简单地对 MATLAB 类和类对象进行了介绍，详细的使用请读者参见相关的参考书籍。

1.2.4 阵列与数组

细心的读者可能已经发现，在前面的讲述过程中，书中将 array 全部解释为阵列，而在其他的一些相关书籍中，一般将 array 翻译为数组。确切地说，这两种解释都是正确的，但是作者在全书中采用了"阵列"这种解释，主要是出于以下原因考虑：

首先，这是为了与一般的计算机高级编程语言中数组的概念相区分。因为在一般的计算机高级编程语言中，数组的定义为一组相同数据类型变量的集合，这里必须非常注意"相同数据类型"，而在 MATLAB 中 array 的概念却没有这种限制，它的元素可以为任意类型变量，如单元阵列，它的每一个元素的类型均可以互不相同；

其次，从数据处理的角度来说，在一般的计算机高级编程语言中，数组虽然是一组相同数据类型的数据的集合，但是在处理时，仍是以数组元素为对象的，即分别对数组中的每个元素进行处理，而在 MATLAB 中却不是这样，所有的处理过程都是以 array 为对象，将 array 视为一个整体，而并非对 array 的每一个元素进行处理（这并不是说 MATLAB 不能对元素进行处理，相反，在 MATLAB 中，提供了丰富的对单个元素进行处理的功能）。这里将 array 解释为阵列也是为了突出整体处理的概念。

1.3 MATLAB 语言程序设计

在 1.2 节中，我们对 MATLAB 使用的数据类型进行了简单的介绍，为的是使读者对 MATLAB 系统有进一步的了解。本节中我们将就 MATLAB 语言的程序设计分四方面进行介绍：

- MATLAB 的基本运算符；
- MATLAB 的基本语句结构；
- MATLAB 控制语句；
- MATLAB M 文件的编写。

1.3.1 MATLAB 的基本运算符

MATLAB 中，针对阵列的计算提供了大量的功能丰富的运算符，包括算术运算符、关系运算符、逻辑运算符以及其他一些特殊运算符，使得对阵列的一些常用操作变得异常的容易。下面我们以矩阵（即二维的数值阵列）为例，对 MATLAB 的基本运算符进行简单地介绍。

1. 算术运算符

在 MATLAB 中，不但提供了完整的传统矩阵算术运算符，还提供了一些新型的矩阵算术运算符，例如左除、右除等，各运算符的使用及功能见表 1.5，不熟悉 MATLAB 的读

者请详细阅读,对以后的MATLAB程序设计大有益处。

表1.5 MATLAB算术运算符

运算符名及使用格式	功 能
plus(A,B) 或 A+B	计算矩阵A与矩阵B的和,要求矩阵A、矩阵B具有相同的维数,或者其中之一为数量,这时相当于矩阵的每一元素与数量相加
minus(A,B) 或 A-B	计算矩阵A与矩阵B的差,要求矩阵A、矩阵B具有相同的维数,或者其中之一为数量,这时相当于矩阵的每一元素与数量相减
mtimes(A,B) 或 A*B	计算矩阵A与矩阵B的数学意义上的乘积,A、B也可以为数量或向量
times(A,B) 或 A.*B	矩阵A与矩阵B的对应元素相乘,要求矩阵A、矩阵B具有相同的维数,或者其中之一为数量,这时相当于矩阵的每一元素与数量相乘
mpower(A,B) 或 A^B	当A与B均为数量时,表示数量A的B次方幂;当A为方阵,B为正整数时,计算方阵A的B次乘积,当B为负整数时,计算方阵A的逆的B次乘积,而B为非整数时,A^B=V* $\begin{bmatrix}\lambda_1^B & & \\ & \ddots & \\ & & \lambda_n^B\end{bmatrix}$ /V,其中V为特征向量矩阵,$\lambda_1,\cdots,\lambda_n$ 为矩阵A的特征值。当A为数量,B为方阵时,A^B=V* $\begin{bmatrix}A^{\lambda_1} & & \\ & \ddots & \\ & & A^{\lambda_N}\end{bmatrix}$ /V,其中 $\lambda_1,\cdots,\lambda_n$ 为矩阵A的特征值,V为相应的特征向量矩阵。当A和B均为矩阵时,无定义
power(A,B) 或 A.^B	相当于计算[A(i,j)^B(i,j)],要求矩阵A与矩阵B为相同维数
mldivide(A,B) 或 A\B	即方程A*X=B的解X
mrdivide(A,B) 或 A/B	即方程X*A=B的解X
ldivide(A,B) 或 A.\B	矩阵B中的元素除以矩阵A中相对应的元素,要求矩阵A、矩阵B具有相同的维数,或者其中之一为数量
rdivide(A,B) 或 A./B	矩阵A中的元素除以矩阵B中相对应的元素,要求矩阵A、矩阵B具有相同的维数,或者其中之一为数量

2. 关系运算符

MATLAB中共提供了六种关系运算符,用于比较两个相同维数的矩阵,它们分别是<(小于),<=(小于或等于),>(大于),>=(大于或等于),==(等于)和~=(不等于),使用它们可以检查矩阵中的元素是否满足某些条件。在MATLAB中,用1表示真,0表示假,这一点与C语言类似。

3. 逻辑运算符

MATLAB提供了六种逻辑运算符(详见表1.6):

表 1.6 逻辑运算符

运算符名及使用格式	功　　能
and(A,B) 或 A&B	矩阵与:结果为 0-1 矩阵,当矩阵 A、B 相对应的元素均为非零值时,结果为 1,否则为 0;要求矩阵 A、矩阵 B 具有相同的维数,或者其中之一为数量
or(A,B) 或 A|B	矩阵或:结果为 0-1 矩阵,当矩阵 A、B 相对应的元素有一个为非零值时,结果为 1,否则为 0;要求矩阵 A、矩阵 B 具有相同的维数,或者其中之一为数量
not(A) 或 ~A	矩阵非:结果为 0-1 矩阵,即对矩阵 A 的元素取反,当 A 的元素为 0 时,结果为 1;元素为 1 时,结果为 0
xor(A,B)	矩阵异或:结果为 0-1 矩阵,当矩阵 A、B 相对应的元素取不同值时,结果为 1,否则为 0;要求矩阵 A、矩阵 B 具有相同的维数,或者其中之一为数量
any(A)	当矩阵 A 中存在任意一个非零元素时,结果为 1,否则为 0
all(A)	当矩阵 A 中所有的元素均为非零值时,结果为 1,否则为 0

此外,MATLAB 还提供了相当多的与逻辑运算相关的内建函数,如 find、finite 等,由于篇幅关系,请读者自行参阅联机帮助。

4. 特殊运算符

除了上述三类较为常见的运算符外,MATLAB 还提供了大量的特殊运算符,包括位运算符、集合运算以及特殊符号运算符,极大地方便了 MATLAB 的使用,表 1.7 列出了其中一部分较为常用的运算符。

表 1.7 特殊运算符(部分)

运算符名及使用格式	功　　能
:	一般用于生成向量、也可用于矩阵行列或矩阵块的表示,例如 1:5 将生成向量[1,2,3,4,5];A(:,j)表示矩阵 A 的第 j 列
%	用于注释
...	行连接符
[ ]	由于生成矩阵
A.'	表示矩阵 A 的转置
bitand(A,B)	对矩阵 A、B 相对应的元素进行按位与计算,要求 A 和 B 为具有相同维数的正整数矩阵,或者其中之一为正整数
bitor(A,B)	对矩阵 A、B 相对应的元素进行按位或计算,要求 A 和 B 为具有相同维数的正整数矩阵,或者其中之一为正整数
union(A,B)	并集(详细使用见联机帮助)
intersect(A,B)	交集(详细使用见联机帮助)

1.3.2 MATLAB 的基本语句结构

MATLAB 语言本质上是一种解释性的语言，用户不但可以通过它直接在 MATLAB 命令提示符下键入语句执行，而且可以使用它编写这样或那样的应用程序，然后回到 MATLAB 环境中，由 MATLAB 解释器进行翻译执行，最后返回处理结果，非常类似于古老的 DOS 操作环境，不过在 MATLAB 命令提示符下，所有符合 MATLAB 语言语法的语句，MATLAB 均认为它们是合法的命令，不受命令个数的约束。

在 MATLAB 语言中，最基本的语句结构为赋值语句，除控制语句外的所有语句都是赋值语句，只不过是存在形式上的复杂和简单差异。基本的赋值语句结构可以表示为如下形式：

变量名列表 = 表达式

其中等号右边为表达式的定义，它可以是 MATLAB 所允许的各种运算或函数调用，等号左边的变量名列表为等号右边 MATLAB 表达式的计算返回值。这里有三点需要说明：

第一，与 C 语言类似，MATLAB 语言敏感字母的大小写(case-sensitive)，例如变量名 dfg 和变量名 DFG 表示的就是不同的变量，在编程时必须非常注意。此外，在 MATLAB 中，基本上所有的内建函数均以小写形式进行声明，如果在 MATLAB 命令提示符下键入如下命令：

```
? a = COS(1)
```

MATLAB 将返回如下错误提示：

```
?? Undefined variable or capitalized internal function COS; Caps Lock may be on.
```

所以在调用 MATLAB 内建函数必须同样小心；

第二，等号右边的表达式可以用多种符号来表示结束，分别代表了不同的含义。当用分号结束时，则左边的变量的结果不会回显在屏幕上；当用逗号或不用任何符号结束时，回车后左边变量的计算结果将显示在屏幕上；当用行连接符“...”结束表达式并回车，MATLAB 将不作任何反应，继续等待用户输入表达式，因为行连接符“...”表示输入未完成，换行后继续输入；

第三，与 C 语言不同，但与 FORTRAN 语言相同，在函数调用时 MATLAB 允许一次返回多个结果，这时赋值语句左边的变量必须用中括号[ ]括起来，例如命令

```
[BW, THRESH] = edge(im, 'Soble')
```

调用了图像工具箱中的边缘计算函数 edge()，其中 im 为图像数据，'Soble' 表示使用 Soble 算子进行边缘计算，返回的边缘图像存放在变量 BW 中，而 THRESH 为计算边缘时使用的边缘域值。

1.3.3 MATLAB 控制语句

与其他计算机高级编程语言类似，MATLAB 提供了完备的控制语句，例如条件转移语句、循环语句等常用的控制语句，从而使 MATLAB 的程序设计变得非常灵活。下面分别加以介绍。

1. while 循环语句

while 循环语句是 MATLAB 提供的两种循环语句中的一种，它的组成结构为：

```
while (条件表达式)
    循环体语句组
end
```

其执行过程为首先判断条件表达式的真假，若为假，则跳出循环体，执行 end 语句后的语句；若为真，则执行循环体中的语句组，执行完毕后，返回条件表达式，重复以上过程，直至条件表达式不再成立时为止，其结构框图见图 1.7。

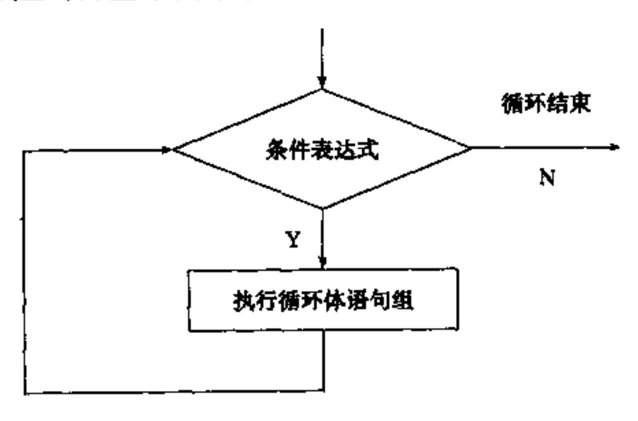


图1.7　while循环结构框图

如果用户希望计算 1 至 100 的累加和，在不使用内建函数的前提下，可以由以下程序段实现：

```
n = 1; s = 0;
while (n<=100),
    s = s + n; n = n + 1;
end
```

在 MATLAB 语言中，允许 while 循环的嵌套。

2. for 循环语句

for 循环语句是 MATLAB 提供的两种循环语句中的另一种，它的组成结构为：

```
for 循环变量 = 表达式1 : 表达式2 : 表达式3
    循环体语句组
end
```

其中表达式 1 的计算结果为循环变量的初始值，表达式 3 的计算结果为循环变量的终止值，而表达式 2 的计算结果为循环变量的步进值，在缺省的情况下，默认步进值为 1。for 循环语句一般用于循环次数已知的情况下。

整个循环的执行过程为：首先初始化，计算表达式 1 和表达式 3 的值，并将表达式 1 的值赋给循环变量；然后进入循环体，首先判断循环变量的取值是否位于表达式 1 和表达式 3 所限定的范围之内，若不在则跳出循环体，若在则执行循环体语句组，之后将循环变

量增加由表达式 2 确定的步进值,然后重复执行循环体内容,直至循环变量不再满足条件为止,结构框图见图 1.8。

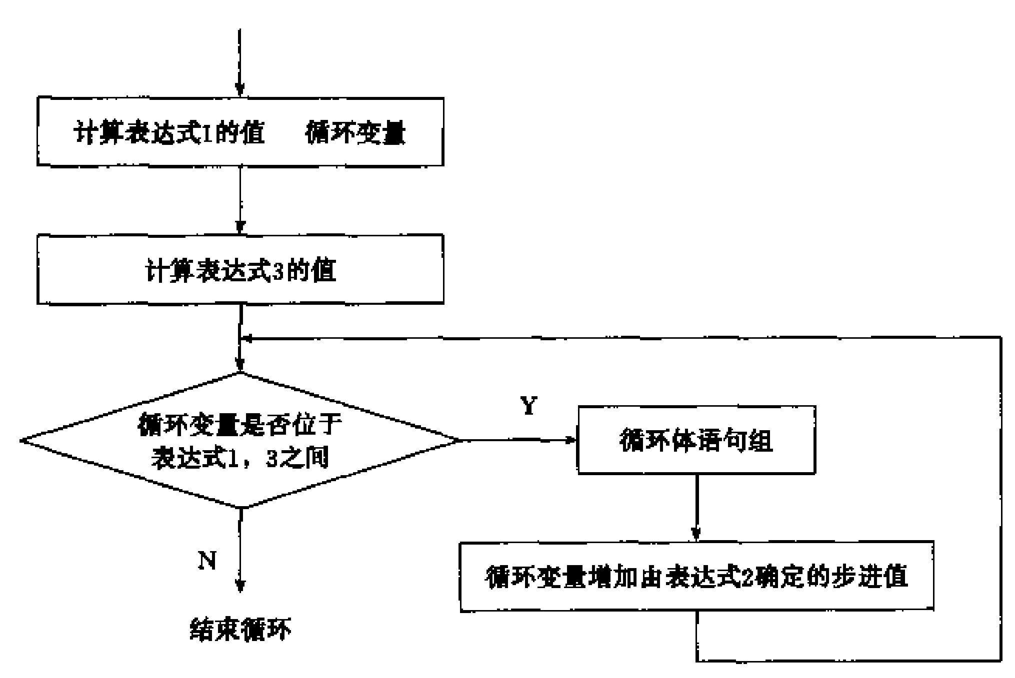


图1.8 for循环语句结构框图

用 for 循环实现 1 至 100 的累加和,可以表述为如下形式:

```
s = 0;
for n = 1 : 1 : 100
    s = s + n;
end
```

同样在 MATLAB 语言中,允许 for 循环的嵌套。

3. if 条件转移语句

if 条件转移语句是 MATLAB 语言提供两种条件转移语句中的一种,它有三种常用的组成结构:

```
if 条件表达式 1          if 条件表达式 1          if 条件表达式 1
    语句组 1                 语句组 1                 语句组 1
end                      else                     elseif 条件表达式 2
                             语句组 2                 语句组 2
                         end                          ……
                                                  end
```

它们一般的执行过程为:首先判断条件表达式 1 的成立是否成立,若成立则执行语句组 1,若不成立,则分为三种情况,第一种情况跳出 if 语句,第二种情况执行 else 后的语句组 2,第三种情况下则先判断条件表达式 2 的成立与否,然后按上面的流程继续向下执行,直至跳出 if 条件转移语句。它们的结构框图分别见图 1.9 的(a)、(b)和(c)。

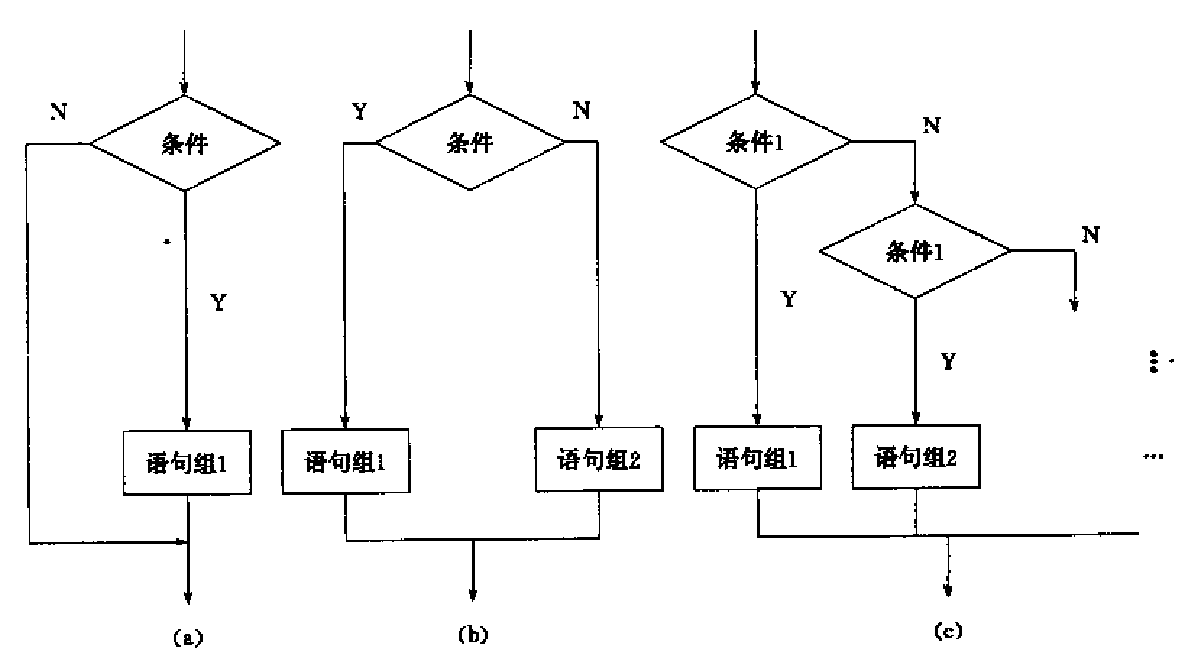


图1.9 if条件转移语句结构框图

例如,用户希望构造一个矩阵 $A=\begin{bmatrix}2 & 1 & & & 0\\ 1 & 2 & 1 & & \\ & 1 & 2 & \ddots & \\ & & \ddots & \ddots & 1\\ 0 & & & 1 & 2\end{bmatrix}$,可以用如下的循环语句来完成:

```
if   I == J
     A(I,J) = 2;
elseif   abs(I-J) == 1
     A(I,J) = -1;
else
     A(I,J) = 0;
end
```

4. switch…case 多重条件转移语句

switch…case 多重条件转移语句是 MATLAB 语言提供两种条件转移语句中的另一种,它的基本组成结构为:

```
switch 选择表达式
    case 常量 1,
        语句组 1
    case {常量 2,常量 3}
        语句组 2
    ……
    otherwise,
        语句组 n +1
end
```

其执行过程为首先计算选择表达式的值，并与各 case 语句中的常量比较，然后选择第一个与之匹配的 case 语句组执行，执行完毕后立即退出 switch 语句组；若没有与选择表达式值相匹配的 case 语句，则执行 otherwise 后的语句组 n+1，并退出 switch 语句组。这里必须注意一点，在一个 case 语句后，可以拥有多个常量表达式。

它与 C 语言的 switch 语句相比存在明显的不同，在 C 语言中，当执行完某个 case 语句组后，如果不使用跳出语句，将继续执行后面的 case 语句，直至 switch 语句结束，而在 MATLAB 语言中，不存在这个问题，当程序执行完一个 case 语句组后即跳出 switch 语句，执行后续的语句。下面是一个典型的 switch 语句例子。

```
switch lower(METHOD)
    case {'linear', 'bilinear'},
        disp('Method is linear')
    case 'cubic',
        disp('Method is cubic')
    case 'nearest',
        disp('Method is nearest')
    otherwise,
        disp('Unknown method.')
end
```

5. break 语句

break 语句是 MATLAB 语言提供的一种非常有用的控制语句，主要用于终止当前正在执行的 for 循环语句和 while 循环语句，以跳出循环体，结束不必要的计算。当 break 语句用于循环嵌套中时，它仅仅中断最内层包含该语句的循环体的执行，而不影响高层循环体的执行。例如以下形式的程序段中：

```
while 条件表达式 1
    for n = a : b : c
        语句组 2
        if 条件表达式 2
            break;
        end
    end
    语句组 2
end
```

当条件表达式 2 为真时，执行 break 语句，程序将跳出内部的 for 循环体，终止其执行，而外部的 while 循环将不受影响继续执行，直到条件表达式 1 得不到满足时，才终止 while 循环的执行。

1.3.4 MATLAB M 文件的编写

由 MATLAB 语言编写而成的文件，习惯上称之为 M 文件，其后缀均为 m。从总体上来说，可以将 M 文件分为两类，即函数 M 文件(Function M-file)和脚本 M 文件(Script M-file)。二者之间存在着较大的差别：

首先，函数 M 文件可以从用户处接受一定数量的输入参数，并返回若干输出参数，而脚本 M 文件一般不接受任何输入参数，也不返回任何的输出参数；

其次，在默认情况下，函数 M 文件中所定义和包含的变量均为局部变量，仅仅在函数执行期间有效，当退出该函数后，所有这些变量全部失效，在 MATLAB 工作环境中，使用 whos 命令看不到它们的存在；而脚本 M 文件却不同，它一般是针对 MATLAB 工作空间中的数据进行操作，当脚本 M 文件执行完毕后，工作空间中的变量将仍然存在，除非在脚本 M 文件中显式删除；

第三，函数 M 文件和脚本 M 文件的用途不尽相同，一般脚本 M 文件用来存放用户需要重复执行的一系列操作，以避免重复地键入大量相同命令，相当于 DOS 操作系统中的批处理命令；而函数 M 文件则一般用来完成某种特定的功能，是用户应用程序的组成部分和 MATLAB 功能的扩展，与 C 语言中的子函数和 FORTRAN 语言中的子例行程序类似，不同的是函数 M 文件可以独立执行；

第四，函数 M 文件和脚本 M 文件的编写格式不尽相同，下面将分别予以介绍。

1. 函数 M 文件的编写

程序段

```
function y = average(x)
% AVERAGE Mean of vector elements.
% AVERAGE(X), where X is a vector, is the mean of vector elements.
% Non-vector input results in an error.
    [m, n] = size(x);
    if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
        error('Input must be a vector')
    end
y = sum(x)/length(x); % Actual computation
```

是一个非常简单但又非常完备的函数 M 文件的定义过程，这里简单是指其仅仅包含五行实际用于判断和计算的语句，而完备是指其具备了所有 MATLAB 函数所共同具有的组成部分。下面将以它为例对函数 M 文件的编写进行讲解。

该函数段定义了一个名为 average、具有一个输入参数和一个输出参数的函数，其功能为计算一个向量元素的平均值。它可以分为五个部分：

第一部分为程序段的第一行，称为函数定义行(function definition line)，它定义了函数的名称 average，输入形式参数 x 和输出形式参数 y，其中 function 为关键字；当函数存在多个返回参数时，需要将这些参数用中括号[ ]包括，并且使用逗号分隔，同时函数可以接受多个输入参数，例如

function [A_1, A_2, A_3,…, A_n] = func _name (B_1, B_2,…, B_m)

此外，在 MATLAB 中还定义了两个非常有用的关键字 nargin 和 nargout，它们分别用来纪录函数调用语句中输入和输出实际参数的个数，通过这种方法，MATLAB 就允许函数调用语句中的实际参数个数可以与函数 M 文件中的形式参数个数不同，并且可以在函数 M 文件中分别针对不同的参数个数进行不同的处理，从而极大地增加了 MATLAB

语言程序设计的灵活性。

第二部分为程序段的第二行，称为 Help 1 line，简称为 H1，意思是指该行为帮助信息的第一行，它的主要作用是当用户检索或者使用 help 命令查询整个目录而非查询单个命令时，MATLAB 系统将显示该行内容，用于函数功能的说明。例如使用 help 命令查询函数 average 所在的目录，假设该目录下仅有函数 average 和另一个名为 allnodes 的小波工具箱函数，则 MATLAB 将显示如下结果：

```
? help \目录名
ALLNODES Tree nodes.
AVERAGE Mean of vector elements.
```

第三部分为程序段的第三、第四行，称为帮助信息(Help Text)，当用户使用 help 命令直接查询该函数时，MATLAB 系统将连同 H1 行将帮助信息完整地显示在屏幕上。例如：

```
? help average
AVERAGE Mean of vector elements.
        AVERAGE(X), where X is a vector, is the mean of vector elements.
Non-vector input results in an error.
```

第四部分为程序段的第五至第九行，称为函数体(function body)，这部分内容是函数的主体，包含了函数的全部计算代码，由它来完成所设计的功能。

第五部分为注释(comment)，它一般由%开始，位于某行代码的前一行或紧随其后，主要用于说明该行代码的功能，程序段中见第九行的后部。

以上五部分中，第一部分和第四部分必不可少，其余三个部分可以省略，但作者强烈建议按格式完整书写，有利于增强程序的可读性和今后的改进。

2. 脚本 M 函数的编写

脚本是 MATLAB M 文件中最简单的一种类型，它的编写也极为简单，基本上没有任何格式上的约束，整个文件可以分为两个部分，即执行语句部分和注释部分，所有注释内容均以符号%为开头，不用定义输入参数和输出参数，不用定义函数名，其存盘时所使用的文件名即将来文件调用时所使用的命令名，但是作者强烈建议书写足够的注释，以便于文件的阅读。

程序段

```
% An M-file script to produce "flower petal" plots
  theta = -pi:0.01:pi;
  rho(1,:) = 2 * sin(5 * theta).^2;
  rho(2,:) = cos(10 * theta).^3;
  rho(3,:) = sin(theta).^2;
  rho(4,:) = 5 * cos(3.5 * theta).^3;
  for i = 1:4
      polar(theta, rho(i,:))
      pause
end
```

为一个比较简单但非常有趣的脚本 M 文件，用于实现一定的绘图功能，有兴趣的读者可以录入并观看运行结果。

1.4　基本的 MATLAB 矩阵操作

在 MATLAB 系统中，阵列是 MATLAB 惟一能够进行处理的对象，而矩阵是阵列的一种具体形式。在非正式的情况下，矩阵和阵列的概念可以互换，但是准确地说，矩阵是一种二维的实数或复数阵列。最初开发和设计 MATLAB 的动机是为了简化复杂的矩阵运算，正是由于这个原因，虽然 MATLAB 历经了二十年的发展，功能得到不断的完善和发展，应用的领域也得到了极大的拓广，版本也由最初的 V1.0 升至现在的 V5.3，但是一直以来，MATLAB 都以矩阵为其最基本的处理对象，为矩阵的操作提供了大量功能丰富、涵盖面广的内建函数，使得对矩阵的操作得到全面的简化，本节将对 MATLAB 中一些基本的矩阵操作如矩阵的构造、矩阵的数学计算进行简单的介绍。

1.4.1　矩阵的构造

矩阵的构造是完成矩阵操作的基本前提，在 MATLAB 中，系统为用户提供了若干种矩阵构造方式，可以用于构造各种不同类型的矩阵，本小节将对其中一些较为常用的构造方式进行介绍。

1. 矩阵构造算符[ ]

使用矩阵构造算符[ ]来进行矩阵的构造极为直观，尤其是在矩阵维数较低的情况下，显得非常方便，只需将所需要定义的矩阵的所有元素按行的顺序排列后，用算符[ ]括起来，并且同一行内的元素用空格或逗号分隔，不同的矩阵行用分号进行分隔即可。例如用户希望定义一个矩阵 $A=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$，按如下形式书写即可：

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9 ]
```

这里必须非常注意一点，矩阵的定义方式与存储方式截然不同，前者为按行定义，而后者按列存储。此外，用户还可以利用算符[ ]构造一种极为特别的矩阵——空矩阵，它不包含任何元素，维数为 0×0，其主要用途是用来传播空矩阵。利用这个特性，可以对矩阵的部分行或列进行删除，例如命令 A(:, 1:3) = [ ]的功能就是将矩阵 A 的第一和第二列删除。

2. zeros、ones 和 eye

函数 zeros 用来构造任意维数的全零矩阵，其使用格式为

```
A = zeros(m ,n) 或 A = zeros(n)
```

其中前一种格式用于构造 m×n 维的全零矩阵，而后一种格式用于构造 n×n 维的全零矩阵。

函数 ones 的使用方式与 zeros 相同，只不过构造的是全一矩阵。

函数 eye 用于构造对角线元素全一而其余元素全零的单位矩阵，使用格式同 zeros。

3. 特殊矩阵构造函数

MATLAB 中提供了大量的特殊矩阵构造函数，极大地方便了矩阵的构造，表 1.8 为部分函数的列表。

表 1.8 特殊矩阵构造函数(部分)

函数名	功能
rand	用于构造元素为一致分布的随机矩阵
randn	用于构造元素为标准正态分布的随机矩阵
hilb	用于构造 Hilbert 矩阵
invhilb	用于构造逆 Hilbert 矩阵
pascal	用于构造杨辉三角形矩阵
vander	用于构造范德蒙矩阵
company	用于构造多项式的伴随矩阵
hadamrd	用于构造哈达马矩阵

1.4.2 矩阵的数学计算

MATLAB 系统不但提供了最基本的矩阵的加、减、乘、除、乘方、转置、逻辑以及关系等运算(详见 1.3.1 节)，还提供了大量的与线性代数紧密相关的数学计算功能，包括矩阵特征值和特征向量的计算、矩阵数字特征值的计算、矩阵的分解等。

1. 特征值和特征向量

在信号处理和模式识别中，方阵的特征值和特征向量是一种非常有用的矩阵信息，对于数据的压缩和特征模式的选择具有重要意义。MATLAB 的内建函数 eig 为这方面的计算提供了完善的功能，其使用格式为：

lambda = eig(A) 或 [V, D] = eig(A)

其中 A 为一个待计算的方阵，lambda 为特征值向量，V 为特征向量矩阵，它的每一列为一个特征向量，D 为一个对角线矩阵，对角线上的元素为与 V 中特征向量相对应的特征值。例如存在方阵 $A=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$，在 MATLAB 命令提示符下键人如下命令：

```
? [V, D] = eig(A)
```

回车后 MATLAB 将显示下面的结果：

```
V =
    0.2320     0.7858     0.4082
    0.5253     0.0868    -0.8165
    0.8187    -0.6123     0.4082
```

```
D =
   16.1168         0         0
         0   -1.1168         0
         0         0   -0.0000
```

2. 矩阵的数字特征

矩阵的数字特征主要是指矩阵的范数和矩阵的秩。在 MATLAB 中矩阵范数的计算由函数 norm 来完成，其格式为

```
p = norm(A, n)
```

其中 A 为任意一个矩阵，n 为范数的阶数，取值为 1，2 或∞，在省略情况下，默认值为 2，p 为计算所得的范数值。对于矩阵 $B=\begin{bmatrix}9 & 11 & 3\\ 4 & 7 & 8\\ 5 & 0 & 12\end{bmatrix}$，分别计算三种范数可得

```
[norm(B,1),norm(B,2),norm(B, inf)]
ans =
    23.0000    19.8692    23.0000
```

其中 inf 为 MATLAB 的关键字，代表∞。

矩阵的秩为矩阵线性无关的行或列的最大数目，可以用函数 rank 进行计算，用法非常简单，对于上面的矩阵 B，使用语句 rank(B) 即可求得秩为 3。使用函数 orth 可以求得相应的正交基。

3. 矩阵的分解

MATLAB 提供了三种常用的矩阵分解方法，分别是三角分解、正交分解和奇异值分解。三角分解是最基本的一种矩阵分解方式，它将一个矩阵分解为一个上三角矩阵和一个下三角矩阵的乘积，因此也称为 LU 分解。MATLAB 中，用于实现三角分解的函数为 lu，其使用格式为

```
[L, U] = lu(A)
```

其中 A 为任意矩阵，U 和 L 分别为上、下三角矩阵，对于矩阵 $C=\begin{bmatrix}7 & 10 & 1\\ 3 & 8 & 0\\ 9 & 4 & 6\end{bmatrix}$，分解后可得到

```
L =
    0.1111    1.0000         0
    0.3333    0.6977    1.0000
    1.0000         0         0
U =
    9.0000    4.0000    6.0000
         0    9.5556    0.3333
         0         0   -2.2326
```

其中 L 可以重新排列为一个对角线元素全为 1 的下三角阵。

正交分解可以将一个方阵或长方阵分解为一个正交矩阵和一个上三角矩阵的乘积，又称为 QR 分解。MATLAB 中，用于实现正交分解的函数为 qr，其使用格式为

```
[Q, R] = qr(A)
```

其中 A 为一个方阵或长方阵，Q 为正交矩阵，R 为上三角矩阵。对于矩阵 C，分解后结果为

```
Q =
   -0.1048   -0.8218   -0.5600
   -0.3145   -0.5068    0.8027
   -0.9435    0.2602   -0.2053
R =
   -9.5394   -7.3380   -5.7656
         0  -11.2318    0.7397
         0         0   -1.7920
```

奇异值分解又称为 SVD 分解，MATLAB 中，用于实现该分解的函数为 svd，其使用格式为

```
[S, V, D] = svd(A)
```

其中 A 为任意矩阵，矩阵 S 和矩阵 D 为正交矩阵，矩阵 V 为对角矩阵。对矩阵 C 进行分解结果为

```
S =
    0.5840    0.5453   -0.6013
    0.5250    0.3113    0.7922
    0.6191   -0.7783   -0.1045
V =
   15.3269         0         0
         0    8.4186         0
         0         0    1.4880
D =
    0.5044   -0.6564    0.5610
    0.8166    0.5737   -0.0630
    0.2805   -0.4899   -0.8254
```

本章分为四节对 MATLAB 作了较为全面的介绍，以使不太熟悉 MATLAB 的读者对 MATLAB 系统有一个整体认识，为下一步内容的学习做好铺垫。由于此部分内容并非本书的主要内容，所以内容较浅，并且有相当多的 MATLAB 精彩内容没有介绍，如 Simulink、图形系统等等，有兴趣的读者可以自行参阅相关书籍进一步学习。从第二章起，将开始全面讲解 MATLAB 应用程序接口的使用及编程。

第 2 章　MATLAB 应用程序接口概述

由第 1 章 MATLAB 系统的简单介绍可以知道，MATLAB 系统是一个功能完善的、自包容的程序设计和数据处理集成环境，使用它所提供的功能、内建函数以及大量的工具箱，几乎可以完成所有的任务，并且无需借助外界的帮助，是一个完全独立的系统。但是如果仅仅如此的话，MATLAB 系统仍将是一个不友好的系统，这是因为：

首先，在 MATLAB 环境中，用户将无法调用外部大量已经用 C 语言或 FORTRAN 语言编写完成的算法，而必须使用 MATLAB 语言进行重新编写，这对于规模较大的程序来说，无疑需要花费大量的时间和精力；

其次，与其他高级计算机编程语言，如 C 语言和 FORTRAN 语言相比较，MATLAB 语言的执行效率较为低下，在进行大规模的数值计算和分析时，MATLAB 显得有些力不从心，有必要借助其他高级语言来进行加速；

第三，MATLAB 系统拥有自己的数据文件格式，而且对于不同的硬件平台和操作系统，文件格式略有差异，这样无疑增大了 MATLAB 系统与其他环境进行数据交换的难度；

第四，在其他的应用程序中，无法调用 MATLAB 系统提供的丰富的函数，从而造成资源的极大浪费。

正是由于这些原因，MATLAB 系统提供了一个非常重要的组件——MATLAB 应用程序接口(MATLAB Application Program Interface)来解决这些问题。它是一个功能完善的接口函数库，通过它可以完成以下功能：

- 在 MATLAB 环境中调用 C 语言或 FORTRAN 语言编写的程序，以提高数据处理的效率；
- 向 MATLAB 环境传送数据或从 MATLAB 环境接收数据，即实现 MATLAB 系统同外部环境的数据交换；
- 在 MATLAB 和其他应用程序间建立客户机/服务器关系，将 MATLAB 作为一个计算引擎，在其他应用程序中调用，从而降低程序设计的工作量。

MATLAB 应用程序接口主要包括三部分内容，分别为 MEX 文件——外部程序调用接口，MAT 文件应用程序——数据输入输出接口和 MATLAB 计算引擎函数库。本章中将对它们进行总体上的介绍。

2.1　MATLAB MEX 文件介绍

MATLAB MEX 文件是 MATLAB 系统的外部程序调用接口。通过它，用户可以完成以下功能：

第一，可以在 MATLAB 系统中像调用 MATLAB 的内建函数一样调用已经存在的用 C 语言和 FORTRAN 语言编写完成的算法，而无须将这些程序重新编写为 MATLAB

的 M 文件，从而使资源得到充分利用；

第二，当使用 MATLAB 进行大规模的数据处理时，MATLAB 往往由于执行效率的问题而显得力不从心，这时可以使用其他高级编程语言进行算法的设计，然后在 MATLAB 中调用，从而大大地提高数据处理的效率；

第三，通过 MEX 文件，用户可以直接对硬件进行编程，进一步拓展了 MATLAB 的应用领域。

可见，MEX 文件——MATLAB 系统外部程序调用接口的功能是相当强大的，通过它不但可以大量地缩短编程时间，并且使 MATLAB 的功能进一步增强。本节中，将对 MEX 文件的一些基础知识进行介绍。

2.1.1　MEX 文件概念

MEX 文件是一种按一定格式，使用 C 语言或 FOTRAN 语言编写的，由 MATLAB 解释器自动调用并执行的动态链接函数，在 Microsoft Windows 操作系统中，这种文件类型的后缀名为 dll(dynamic link library)。而在其他的平台上，则有较大的变化，例如在 Apple 公司的 Macintosh 上，后缀名为 mex，在其他类型的工作站上，mex 文件的后缀名更是大不相同，这里不再赘述。以后在不加说明的情况下，本书所使用的平台均为 Microsoft Windows 操作系统。

MEX 文件的使用极为方便，只需在 MATLAB 命令提示符下键入 MEX 文件名即可，这与 MATLAB 的内建函数的调用方式完全相同。当用户执行一个 MEX 文件时，MATLAB 系统将首先搜寻 MATLAB 系统的所有可搜寻路径(通过路径浏览器设置)，然后载入并执行第一个与用户键入的文件名相匹配的可执行文件。由于在 MATLAB 中，存在两种类型的可执行文件，即 MEX 文件和 M 文件，如果一个文件名同时存在两种类型的可执行文件该如何呢？MATLAB 系统规定，MEX 文件的执行优先级高于 M 文件，这样 MEX 文件将优先执行。

此外，由于 MEX 文件没有提供应有的帮助信息，一般情况下，每当构造一个 MEX 文件，就应该编写一个 MATLAB 的 M 文件，作为相应的帮助文件，并且存放在同一个目录下。在该文件中，不包含任何的可执行语句，只是包含一些帮助信息，用来对同名的 MEX 文件的功能及输入输出参数进行说明，这样，用户就可以在 MATLAB 提示符下，通过使用 help 命令获取帮助了。因为在 MATLAB 解释器中，help 命令仅仅对与命令中同名的 M 文件进行查找，而忽略对 MEX 文件的查找。

2.1.2　mx-函数和 mex-函数的区别

在 MATLAB 外部程序接口函数库中，存在两种类型的库函数，分别以 mx 和 mex 为前缀，并且分别完成不同的功能。

1. mx-函数库

mx-函数库是 MATLAB 外部程序接口函数库中提供的一系列函数，它们均以 mx 为前缀，主要功能是为用户提供了一种在 C 语言和 FORTRAN 语言中创建、访问、操作和删除 mxArray 结构体对象的方法，例如函数 mxGetPi，其功能就是获取 mxArray 结构体

对象虚数部分的指针。

相关于 C 语言中所有 mx-函数均在目录

<MATLAB 根目录>\EXTERN\INCLUDE

下的头文件 MATRIX.H 中得到声明。在后面的章节中，将对所有的这些函数进行详细的解释和说明。

FORTRAN 语言的 mx-函数无论是在定义还是在使用上与 C 语言的 mx-函数均存在较大的不同，在后面的章节中，将分别讲述，请读者注意区分。

2. mex 函数库

mex-函数库同样是 MATLAB 外部程序接口函数库中提供的一系列函数，它们均以 mex 为前缀，主要功能是与 MATLAB 环境进行交互，从 MATLAB 环境中获取必要的阵列数据，并且返回一定的信息，包括文本提示，数据阵列等。这里必须注意一点，以 mex 为前缀的函数只能用于 MEX 文件之中。

相关于 C 语言中所有 mex-函数均在目录

<MATLAB 根目录>\EXTERN\INCLUDE

下的头文件 MEX.H 中得到声明，在 3.7 节中，将对所有的这些函数进行详细的解释和说明；而与 FORTRAN 语言相关的 mex-函数，在后面讲解 FORTRAN 语言 MEX 文件时，书中将同样予以详细的解释和说明。

2.1.3 MATLAB 阵列在 C 语言中的声明

在学习 MEX 文件的编写前，非常有必要对 MATLAB 阵列(Array)在 C 语言中的声明进行一定的解释，因为在后续的章节中将大量地涉及到此方面的内容。

在 C 语言中，MATLAB 阵列被声明为一个名为 mxArray 的结构体，该结构体的定义被包含在目录

<MATLAB 根目录>\EXTERN\INCLUDE

下的头文件 MATRIX.H 中，其具体定义为：

```
typedef struct mxArray_tag mxArray;
struct mxArray_tag
{
    char     name[mxMAXNAM];
    int      reserved1[2];
    void    *reserved2;
    int      number_of_dims;
    int      nelements_allocated;
    int      reserved3[3];
    union {
        struct
        {
            void   *pdata;
```

```
                void    *pimag_data;
                void    *reserved4;
                int      reserved5[3];
            } number_array;
        }     data;
    };
```

它主要包含了以下几方面的内容：

- 阵列的类型(Classname)，用于标示阵列的数据类型，在头文件 MATRIX.H 中，数据类型被定义为一个拥有 16 种取值的枚举数据类型；
- 阵列的名字(Name)；
- 阵列维数(Dimensions)，用于标示阵列的大小，对于数量、向量以及矩阵，它们的维数都为 2；
- 阵列数据，即阵列实际所包含的内容；
- 当阵列为数值阵列时，标示出阵列为实数阵列还是复数阵列；
- 当阵列为稀疏矩阵时，记录非零元素的索引；
- 当阵列为结构体或对象，记录结构体或对象的域名及域个数。

不同的矩阵数据类型，不同的矩阵存储方式，对结构体 mxArray 的构造有着相当大的影响。例如一个普通的复数矩阵，概念上应该存在四个域，分别为矩阵的行向量数 M、矩阵的列向量数 N、指向矩阵数据实数部分的指针 pr 和指向矩阵数据虚数部分的指针 pi，对于一个实数的普通矩阵，其虚数指针 pi 为空；但是如果当一个矩阵为稀疏矩阵时，仅仅使用以上四个域就不能满足需求了，还需要另外的四个域，分别为矩阵中非零元素个数的上限 nzmax，矩阵中实际存在的非零元素个数 nnz，指向存储矩阵非零元素行索引值的向量的指针 ir 和指向存储矩阵非零元素列索引值的向量的指针 jc。所有这些变量的值均可以通过以 mx 为前缀的函数从结构体 mxArray 中获得，所以如果读者不能很好地理解上面 mxArray 的定义，没有关系，MATLAB 接口函数库已经对 mxArray 结构体进行了很好的封装，无需关心太多的细节。

2.1.4　系统配置

如果用户希望编译生成 MEX 文件，首先必须具备两个最基本的条件，其一是要求已经安装 MATLAB 应用程序接口组件及其相应的工具；其二是要求有合适的 C 语言或 FORTRAN 语言编译器，如果用户是工作在 Microsoft Windows 平台上，那么用户所使用的编译器必须支持 32 位的 Windows 动态链接库(DLL)才能满足要求。

当具备了这两个条件后，也就具备了最基本的编译生成 MEX 文件的基础。不过这样还是不够的，在编译 MEX 文件之前还必须对 MATLAB 系统进行配置，以使 MATLAB 知道用户所使用的编译器类型及其路径。MathWorks 公司的开发者们为了简化配置工作，为命令 mex 提供了参数-setup(仅仅针对 Windows 和 Macintosh)，当用户键入该命令后，只需简单地回答系统的提问，即可完成全部的配置工作。

1. 配置流程

下面将基于 Microsoft Windows NT4.0 操作系统及 MATLAB V5.3，使用 Mi-

crosoft Visual C++编译器，对系统的配置过程逐步进行讲解。首先，在 MATLAB 命令提示符下键入配置命令

```
? mex -setup
```

系统将显示如下内容：

```
Please choose your compiler for building external interface (MEX) files.
Would    you like mex to locate installed compilers [y]/n?
```

提示用户选择 yes 或 no，以确定是否让 mex 命令自动定位用户计算机中已经安装的各类型编译器，选择 y 表示同意后，mex 命令将列出用户计算机中所有 MATLAB 认识的编译器，由于在作者的计算机中，仅安装了 Microsoft Visual C++ 6.0 和 MATLAB V5.3 自带的 Lcc 编译器，所以 mex 的执行结果如下：

```
mex has detected the following compilers on your machine:

[1] : Microsoft 6.0 compiler in e:\Program Files\Microsoft Visual Studio
[2] : Lcc compiler in E:\MATLAB\sys\lcc
[0] : None

Please select a compiler. This compiler will become the default:
```

提示用户选择哪一个编译器，作为默认的 MEX 文件编译器，选择 1 后，mex 命令将进一步提示

```
Please verify your choices:

Compiler: Microsoft 6.0
Location: e:\Program Files\Microsoft Visual Studio

Are these correct? ([y]/n):
```

确认所选择的编译器的路径是否正确，选择 y，mex 命令将对系统进行配置，若成功将显示：

```
The default options file:
"C:\WINNT \Profiles\Administrator\ApplicationData\MathWorks\MATLAB\
            mexopts.bat"is being updated...
```

若不成功将显示：

```
E:\MATLAB\BIN\MEX.BAT: Error: mex requires that the Microsoft Visual C++ 6.0
directories "VC98" and "Common" be located within the same parent directory.
(Could not find e:\Program Files\Microsoft Visual Studio\Common.)
```

以说明出错的原因，例如以上原因表示路径设置不正确，不能找到所需要的编译器的执行文件；当选择 n 时，mex 命令将显示

```
E:\MATLAB\BIN\MEX.BAT: No compiler was set
```

表示没有编译器被设置。

此外，如果用户在第一步选择时，选择了 n，表示不需要 mex 命令自动定位编译器，这时，mex 命令将显示如下编译器列表让用户选择：

```
Choose your C compiler:
[1] Borland C           (version 5.0, 5.2, or 5.3)
[2] Microsoft Visual C  (version 4.2, 5.0, or 6.0)
[3] Watcom C            (version 10.6 or 11)
[4] Lcc C               (C compiler bundled with MATLAB)

Or choose a Fortran compiler:
[5] DIGITAL Visual Fortran (version 5.0)

[0] None

compiler: 2
```

选择 2 之后系统将进一步提示用户，选择所使用的编译器版本，

```
Choose the version of your C compiler:
[1] Microsoft Visual C 4.2
[2] Microsoft Visual C 5.0
[3] Microsoft Visual C 6.0

version: 3
```

选择 3 之后，系统将显示 mex 命令自身所找到的该版本编译器的位置，如下：

```
Your machine has a Microsoft Visual C compiler located at e:\Program Files\Micro
soft Visual Studio.
Do you want to use this compiler [y]/n? n
```

并提示用户确认，不过此时 mex 命令提供的路径与使用自动定位模式时提供的路径相同，不一定正确，因此选择 n，系统将显示：

```
Please enter the location of your C compiler:
        [e:\Program Files\Microsoft Visual Studio]
```

要求用户输入编译器所在的路径，如：

```
e:\Program Files\Microsoft Visual Studio\Vc98
```

回车之后，系统将进行最后的确认：

```
Compiler: Microsoft Visual C 6.0
Location: e:\Program Files\Microsoft Visual Studio\Vc98
Are these correct? ([y]/n): y
```

如果回答 y，系统将显示

```
The default options file:
"C:\WINNT\Profiles\Administrator\Application
Data\MathWorks\MATLAB\mexopts.bat"is being updated...
```

如果回答 n，系统将显示

```
E:\MATLAB\BIN\MEX.BAT: No compiler was set
```

表示系统配置失败，必须重新配置。如果用户使用的是 Windows 98 操作系统或者 MATLAB 系统和编译器位置安装不同，配置过程和内容可能略有不同。

整个配置过程成功结束之后，必须对系统进行进一步验证，以确认配置的正确性。最简单的确认方法就是对一个 MEX 文件的源程序进行编译，看是否能生成可执行的 DLL 文件。在目录

MATLAB 根目录\EXTERN\EXAMPLES\MEX\

中存在许多 MATLAB 提供的范例程序，用户可以对其中任意一个进行编译，以确认配置的正确性。以文件 yprime.c 为例，在 MATLAB 命令提示符下键入命令：

```
? mex yprime
```

若配置成功，回车后 MATLAB 将不显示任何内容直接回到 MATLAB 命令提示符下，表示 DLL 文件已经生成，这时就可以直接在 MATLAB 中执行函数 yprime，得

```
? yprime(1, 1:4)
ans =
    2.0000    8.9685    4.0000    -1.0947
```

若编译不成功，系统将返回一定的出错原因。

对于使用 MATLAB V5.2 的读者来说，在配置过程中差别较大，首先 V5.2 不支持编译器的自动定位，而且在配置过程中，显示的 Visual C++编译器的最高版本为 5.0。不过还要请使用 MATLAB V5.2 版本的读者放心，MATLAB V5.2 同样支持 Visual C++ 6.0，这一点作者已在微机上调试通过。

2. 选项文件(option file)

选项文件是系统配置过程中产生的一个记录配置信息的文件，其后缀为 .bat，是一个批处理文件，在 Windows 操作系统中的资源管理器下，可以通过鼠标左键选取该文件，然后单击鼠标右键，将出一个浮动菜单，选取其中的“编辑”菜单项，即可以打开选项文件，并可以编辑其内容。不过建议不熟悉选项文件的读者尽量不要对文件进行编辑，有可能导致系统配置信息的损坏，不能够正确编译 MEX 文件。

在使用命令

```
mex -setup
```

对系统进行配置时，MATLAB 系统将配置信息写入默认的选项文件，配置成功后，系统将显示

```
Default options file is being updated...
```

表明选项文件已经更新成功。默认的选项文件名为 mexopts.bat，对于 Windows98 操作系统来说，该文件位于目录

Windows 根目录\ Application Data\MathWorks\MATLAB\mexopts.bat

下；而对于 Windows NT 操作系统来说，该文件位于目录

Windows NT 根目录\Profiles\Administrator\Application Data\MathWorks\MATLAB\mexopts.bat

下，并且 MATLAB 应用程序接口中所有相关的默认选项文件均存放在该目录下。

3. 简单的 MEX 文件编译

系统配置完成后，就可以使用命令

mex <源文件名>

对文件进行编译了，该命令使用的是默认的系统信息选项文件 mexopts.bat。同时 MATLAB 系统为命令 mex 提供了参数-f，通过该命令参数，可以让用户使用不同的选项文件对源文件进行编译，命令格式如下：

mex -f <选项文件名> <原文件名>

在 Windows 操作系统上，MATLAB 为各种不同的编译器提供了不同的选项文件，例如 MATLAB 为 Visual C++ 6.0 编译器提供的选项文件名为 msvc60opts.bat，该文件位于目录

MATLAB 根目录\bin

下。相关的编译器选项文件，请读者参阅 2.4 节。

在第三章中，将对 C 语言 MEX 文件的编写以及相应的库函数进行详细的介绍。

在第四章中，将对 FORTRAN 语言 MEX 文件的编写以及相应的库函数进行详细的介绍。

2.2 MATLAB MAT 文件介绍

2.2.1 MAT 文件的概念、格式及功能

MAT 文件是 MATLAB 数据存储默认的文件格式，它的文件名是以 .mat 为后缀结尾。MAT 文件的来源也正是因为这些文件有 .mat 的后缀扩展名。

MAT 文件由文件头、变量名和变量数据三部分组成。其中，MAT 文件的文件头又由以下部分组成：MATLAB 的版本信息、操作平台的信息、文件创建的时间。我们可从文本编辑器程序中打开一个 MAT 文件，查看其文件头中的信息。例如，某 MAT 文件的文件头：5.0 MAT-file, Platform: PCWIN, Created on: Tue Jan 18 16:25:07 2000。变量名是存储在文件中的变量的名字，在给变量取名时，应尽量做到见名知义。变量的数据类型包括 MATLAB 中能够使用到的大部分数据类型，包括字符串、矩阵、多维阵列、结构和单元阵列。MATLAB 以字节流的方式顺序将数据写入到 MAT 文件中去，存储在硬盘上的数据是以二进制的格式保存。

在 MATLAB 中，用户可以直接使用 save 命令存储在当前工作内存区中的数据，把这些数据存储成二进制的 MAT 文件。Load 命令则执行相反的操作，它把磁盘中的 MAT 文件的数据读取到 MATLAB 工作区中。而且 MATLAB 提供了带 mat 前缀的 API 库函

数，使我们能够容易对 MAT 文件进行操作。我们可以轻松地编写能够调用这些库函数的 C 或 FORTRAN 语言的应用程序，用来创建或修改 MAT 文件，完成应用程序与 MATLAB 的数据共享。我们建议读者使用这些库函数，而不是尝试自己去编写这些函数，这样即使将来 MAT 文件的结构发生了改变，用户也不必修改自己的应用程序。

2.2.2 MAT 文件的优势

在 MATLAB 中，用户可以通过多种方法向 MATLAB 输入数据或者从 MATLAB 中获取数据，这些方法在特定的情况下都具有其合理性，有关这些方法在本书的第五章第一节中，我们将进行详细的介绍。但是与其他方法相比，用户将会发现，MAT 文件具有非常独特的优势：

首先，通过使用 MAT 文件来完成应用程序与 MATLAB 之间的数据交换，意味着我们能够利用 MATLAB 提供的便捷机制，高效地完成我们的工作；

其次，即使是在不同的操作平台之间，通过 MAT 文件来完成数据交换也不存在任何问题，因为 MAT 文件是独立于平台之外的。在 MAT 文件中，其数据是由二进制编码组成，并且在 MAT 文件的文件头中还包含了某一种操作平台的符号，当 MATLAB 装载 MAT 文件时，会检查这些符号，假如这些符号表明这个文件是不同操作平台的信息，MATLAB 就会自动进行必要的转换。

2.2.3 系统的配置及 MAT 文件应用程序的编译

使用 MAT 文件不需要经过特别的系统配置，在一般情况下，当对 MEX 文件的系统配置完成之后，对 MAT 文件的系统配置也就基本完成，无需额外的步骤就可以对 MAT 文件程序进行编译了。

对 MAT 文件程序的编译与 MEX 文件的编译略有不同，它没有默认的选项配置文件，在对程序进行编译时，必须使用 mex 命令提供的命令参数-f 来指定适合不同编译器的选项文件，例如 MATLAB 为 Microsoft Visual C++6.0 编译器的选项文件名为 msvc60engmatopts.bat，该文件位于目录

```
MATLAB 根目录\bin
```

下，使用它对 MAT 文件程序进行编译时的命令格式如下：

```
mex -f MATLAB 根目录\bin\msvc60engmatopts.bat matcreat.c
```

如果配置无误的话，这样就可以生成名为 matcreat.exe 的可执行程序，该程序执行时将构造一个名为 mattest.mat 的 MAT 文件。其他一些编译器的选项文件请读者参见 2.4 节。

总体说来，通过 MAT 文件来完成 MATLAB 的数据输出输入任务较为方便和通用，是一种不错的选择。在本书的第五章中，将对 MAT 文件应用程序的编写和库函数的使用进行详细的讲解。

2.3　MATLAB 引擎函数库介绍

2.3.1　MATLAB 引擎的概念及功能

MATLAB 引擎函数库是 MALAB 提供的一系列程序的集合，它允许用户在自己的 C 语言或 FORTRAN 语言应用程序中对 MATLAB 进行调用，将 MATLAB 作为一个计算引擎使用，让其在后台运行，完成复杂的矩阵计算，简化前台用户程序设计的任务。MATLAB 引擎函数库因此而得名。

在用户启动 MATLAB 引擎时，相当于启动了另外一个 MATLAB 进程，其在后台运行。用户应用程序通过 MATLAB 引擎函数库中提供的函数完成与 MATLAB 引擎之间进行数据交换和命令传送的任务。在 UNIX 操作系统上，两者之间的通信是通过管道(PIPES)来完成；而在 Windows 操作系统上，则主要是通过 ActiveX 来完成，这在后面的章节中将详细介绍。

通过 MATLAB 引擎，用户将可以完成以下任务：

- 可以将 MATLAB 作为一个功能强大的和可编程的数学函数库，调用 MATLAB 中大量的数学计算函数，完成复杂的计算任务，例如对一个矩阵进行转置或计算快速傅里叶变换，这对于普通的 C 语言或 FORTRAN 语言编程，将是非常麻烦的，而使用了 MATLAB 计算引擎之后，仅仅几行语句就可以完成任务；
- 可以为一个特定的任务构建一个完整的系统，其中前台的用户图形界面可以使用 C 语言进行定制，从而更加友善和易用，而后台的计算任务可交由 MATLAB 引擎来完成。这种开发模式，将极大地缩短应用程序的开发周期，目前已经有一些较为成功的系统，例如雷达信号分析系统和气象色谱分析系统。

同时，将 MATLAB 作为一个独立的计算进程也有诸多好处：

首先，用户编制的应用程序可以独立于 MATLAB 的解释性执行环境而执行，真正生成独立可执行的应用程序；

其次，在 UNIX 操作系统中，用户不但可以在本地计算机上调用 MATLAB 引擎，而且可以通过网络调用其他计算机上的 MATLAB 引擎，这样可以充分利用网络的资源，将大量的计算任务交给网络上最快的计算机完成；

第三，所有这些功能可以仅仅通过链接一小部分引擎库函数来完成，而无须将所有的 MATLAB 代码链接到用户程序，在很大程度上，缩减了程序文件的长度。

2.3.2　系统的配置及 MATLAB 引擎应用程序的编译

使用 MATLAB 引擎不需要经过特别的系统配置，在一般情况下，当对 MEX 文件的系统配置完成之后，对 MATLAB 引擎的系统配置也就基本完成，无需额外的步骤就可以对 MATLAB 引擎程序进行编译了。

对 MATLAB 引擎程序的编译与 MEX 文件的编译略有不同，它没有默认的选项配置文件，在对程序进行编译时，必须使用 mex 命令提供的参数-f 来指定适合不同编译器的选项文件，例如 MATLAB 为 Microsoft Visual C++6.0 编译器的选项文件名为 msvc60engmatopts.bat，该文件位于目录

MATLAB 根目录\bin

下，使用它对 MATLAB 引擎程序进行编译时的命令格式如下：

```
mex -f MATLAB 根目录\bin\msvc60engmatopts.bat engfilename.c
```

如果配置无误的话，这样就可以生成名为 engfilename.exe 的可执行程序，该程序执行时将在后台开启一个 MATLAB 进程，用于完成计算任务。其他一些编译器的选项文件请读者参见 2.4 节。

总体说来，MATLAB 引擎的功能是相当强大的，在本书的第六章中，将对使用 MATLAB 引擎函数库的编程和库函数的使用进行详细的讲解。

2.4 选项文件说明

MATLAB V5.3 版本为不同的编译器提供了不同的选项文件，在 Widows 操作系统中，这些选项文件均放在目录

MATLAB 根目录\bin

中。本节中，我们将基于 Windows 操作系统，对 MATLAB 系统提供的选项文件进行一定的说明，如果读者使用的是其他类型的操作系统，请参阅相关的联机帮助获取信息。

在 Windows 操作系统中，MATLAB 对一些常用的 C 语言和 FORTRAN 语言编译器提供了全面的支持，尤其是在 V5.3 版本中，内容更加丰富。

2.4.1 C 语言选项文件

C 语言选项文件从总体上可以分为三类，即供编译 MEX 文件使用的选项文件，供编译 MAT 文件和使用 MATLAB 引擎函数的程序的选项文件以及供编译使用 C\C++数学函数库程序的选项文件，在本小节中，将对前两种进行介绍，而最后一种将留到第八章进行介绍。

1. MEX 选项文件

在 MATLAB V5.3 版本中，为三种 C 语言的编译器，即 Borland C++，Visual C++和 Watcom C++提供了不同的选项文件，详见表 2.1。

表 2.1 MEX 选项文件

编译器类型	版 本	选项文件名
Borland C++	5.0	bccopts.bat
	5.3	bcc53pots.bat
Visual C++	4.2	msvcopts.bat
	5.0	msvc50opts.bat
	6.0	msvc60opts.bat
Watcom C++	10.6	watopts.bat
	11	wat11opts.bat

可以看出，MATLAB V5.3 版本几乎对目前所使用的各种类型和各种版本的编译器

都提供了支持，在路径设置正确的前提下，编译 MEX 文件时用户只需按照自己的编译器类型选择相应的选项文件进行编译即可完成一切操作。

对于目前比较流行的另一种编译器 C++ Builder 来说，MATLAB 并没有提供显式的支持，也没有为其提供专用的选项文件，不过请习惯使用C++ Builder 读者放心，完全可以使用 Borland C++的选项文件来配合 C++ Builder 编译器对 MEX 文件进行编译，或者在使用命令 mex -setup 配置默认选项文件时，选择 Borland C++编译器，但将路径指向 C++ Builder，这一点作者已经在微机上调试通过。这可能是因为 C++ Builder 和 Borland C++同为一个公司的产品，所以编译器选项参数相同的原因吧。

2. MAT 和 MATLAB 引擎选项文件

与 MEX 选项文件相同，MAT 和 MATLAB 引擎选项文件也因编译器的类型和版本不同而各异，详见表 2.2。

表 2.2 MAT 和 MATLAB 引擎选项文件

编译器类型	版 本	选项文件名
Borland C++	5.0	bccengmatopts.bat
	5.3	bcc53engmatopts.bat
Visual C++	4.2	msvcengmatopts.bat
	5.0	msvc50engmatopts.bat
	6.0	msvc60engmatopts.bat
Watcom C++	10.6	watengmatopts.bat
	11	wat11engmatopts.bat

同理，除了以上三种类型的编译器以外，可以使用C++ Builder 配合 Borland C++的选项文件来对 MAT 文件和 MATLAB 引擎进行编译。

2.4.2 FORTRAN 语言选项文件

相对于C 语言来说，MATLAB 所提供支持的 FORTRAN 语言的种类和版本就少多了，仅仅提供了对 DIGITAL Visual Fortran V5.0 版本的显式支持，其 MEX 选项文件的文件名为 df50opts.bat，MAT 和 MATLAB 引擎选项文件的文件名则为 df50engmatopts.bat。不过对于使用 Microsoft Fortran PowerStation 的读者来说，使用上面的两个选项文件同样可以使用 Microsoft Fortran PowerStation 编译器对 Fortran 语言的 MEX 文件、MAT 文件和 MATLAB 引擎程序进行编译，这一点作者同样在微机上调试通过。

第 3 章　C 语言 MEX 文件的编写

C 语言 MEX 文件，顾名思义就是基于 C 语言编写的 MEX 文件，是 MATLAB 应用程序接口的一个重要组成部分。通过它不但可以将现有的使用 C 语言编写的函数轻松地引入 MATLAB 环境中使用，避免了重复的程序设计，而且可以使用 C 语言为 MATLAB 定制用于特定目的的函数，以完成在 MATLAB 中不易实现的任务，此外还可以使用 C 语言提高 MATLAB 环境中数据处理的效率。在本章中，我们将首先对 C 语言 MEX 文件的构成及其执行流程进行说明，然后对 C 语言 MEX 文件的编写进行全面的讲解，并给出具体的例子，最后介绍相关的 C 语言 MEX 文件的库函数。

3.1　C 语言 MEX 文件

3.1.1　一个简单的例子

在开始介绍 C 语言 MEX 文件的构成前，先请大家看一个非常简单的样例程序 myplus.c，它定义了两个 double 类型数量间的加法运算。该程序的结构非常清晰，极具代表性，请读者仔细阅读，相关内容将在随后的章节中进行详细讲解。

```
/* 头文件包含 */
#include "mex.h"

void myplus(double y[], double x[], double z[])
{
     y[0] = x[0]+z[0];
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
double *x, *y, *z;
     int mrows0, ncols0;
     int mrows1, ncols1;

     /* 检查输入输出变量的个数 */
     if(nrhs != 2)
          mexErrMsgTxt("Two inputs required.");
     else
          if(nlhs>1)
               mexErrMsgTxt("Too many output arguments");
```

```
    /* 输入的元素必须为两个非复数的 double 类型数量 */
    mrows0 = mxGetM(prhs[0]);
    ncols0 = mxGetN(prhs[0]);
    if ( ! mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
                                   ! (mrows0==1 && ncols0==1) )
        mexErrMsgTxt("Inputs must be all noncomplex scalar double.");
    mrows1 = mxGetM(prhs[1]);
    ncols1 = mxGetN(prhs[1]);
    if ( ! mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
                                   ! (mrows1==1 && ncols1==1) )
        mexErrMsgTxt("Inputs must be all noncomplex scalar double.");
    if (mrows0! =mrows1 || ncols0! =ncols1)
        mexErrMsgTxt("Inputs must be same dimension.");

    /* 为返回参数创建矩阵 */
    plhs[0] = mxCreateDoubleMatrix(mrows0,ncols0, mxREAL);

    /* 为输入输出参数赋值 */
    x = mxGetPr(prhs[0]);
    z = mxGetPr(prhs[1]);
    y = mxGetPr(plhs[0]);

    /* 调用 myplus 函数 */
    myplus(y,x,z);
}
```

3.1.2　C 语言 MEX 文件源程序的构成

仔细阅读过以上程序的读者可能已经发现,C 语言 MEX 文件的源程序主要由两个截然不同的部分组成,它们分工明确,分别用于完成不同的任务:

第一部分称为计算子例行程序(computational routine),它包含了所有实际完成计算功能的源代码,用来完成实际的计算工作;

第二部分称为入口子例行程序(gateway routine),它是计算子例行程序同MATLAB环境之间的接口,用来完成两者之间的通信任务。

入口子例行程序的名字为 mexFunction,拥有四个参数,分别为 prhs、nrhs、plhs 和 nlhs,其中 prhs 为一个 mxArray 结构体类型的指针数组,该数组的数组元素按顺序指向所有的输入参数;nrhs 为整数类型,它标明了输入参数的个数;plhs 同样为一个 mxArray 结构体类型的指针数组,该数组的数组元素按顺序指向所有的输出参数; nlhs 则标明了输出参数的个数,为整数类型。

入口子例行程序的具体的使用格式如下:

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
```

```
    ......
    /* 一些必要的C语言代码,用来完成MATLAB与
    计算子例行程序之间的通信任务 */
}
```

在入口子例行程序中,用户主要可以完成两个方面的任务:一方面,是从输入的 mxArray 结构体中获得计算所需的数据,然后在用户的计算子例行程序中加以使用,例如在 3.1.1 中的例子程序中,通过语句 x = mxGetPr(prhs[0])和语句 y= mxGetPr(plhs[0]),分别从输入阵列中得到计算所需的数据指针 x 和 y;另一方面,用户同样可以将计算完毕的结果返回给一个用于输出的 mxArray 结构体,这样 MATLAB 系统就能够认识从用户计算子例行程序返回的结果。

MEX 源文件的两个组成部分既可以像程序 myplus.c 那样存放在一个文件中,也可以分为两个文件来存放,这无关紧要,重要的是文件中必须对头文件 mex.h 进行包含,因为该头文件中不但包含了最基本的头文件 matrix.h,而且包含了所有以 mex 为前缀的库函数的声明。

3.1.3　C 语言 MEX 文件的执行流程

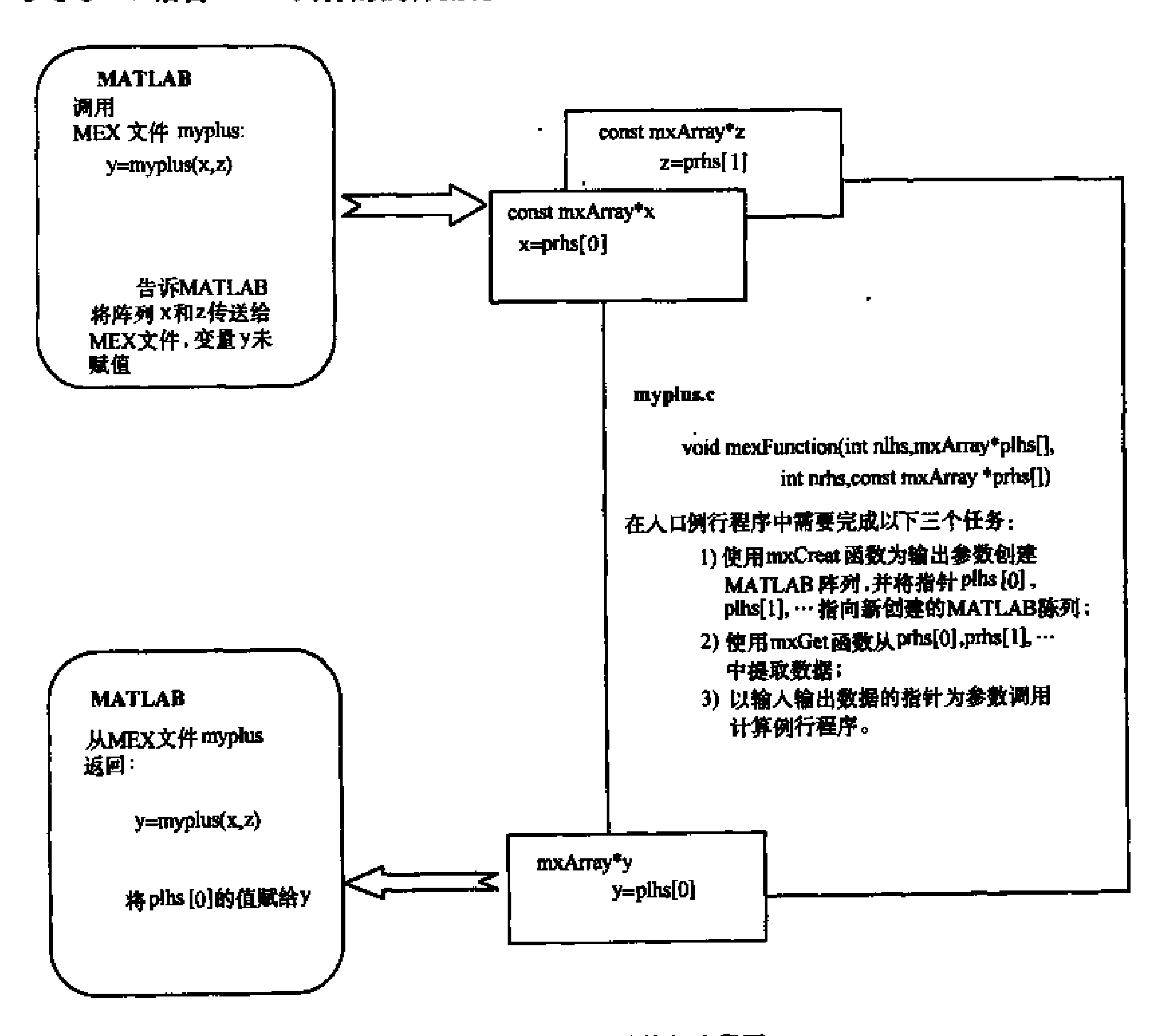


图 3.1　MEX 文件的执行流程图

当对一个 C 语言 MEX 文件的源程序进行编译后(编译的方法见 2.1.4)，如果成功即可以得到与源程序名相同的 DLL 文件，建议将源程序的取名与程序中计算子例行程序的名字保持相同，这样比较直观而且易于使用和记忆。

在 MATLAB 的工作环境中，按照 MATLAB 语言的语法

```
[a, b, c, …] = mexfile_name(x, y, z, …)
```

正确地键入 MEX 文件名和 MEX 文件所需的参数，就可以运行 MEX 文件了。这时，参数 plhs 和参数 prhs 分别为包含所有输出和输入参数指针的 mxArray 结构体类型的指针数组，参数 nlhs 和 nrhs 则分别包含了输出和输入参数的个数。以程序 myplus.c 为例，当对其进行编译后，可以得到文件名为 myplus.dll 的动态链接程序，在 MATLAB 命令提示符下键入命令

```
y = myplus(x, z)
```

之后，MATLAB 解释器将首先对各参数进行赋值，如下：

```
nlhs = 1;             nrhs = 2;
plhs → NULL;          prhs → (y, z) ;符号"→"表示指向
```

然后调用入口子例行程序 mexFunction，来完成计算任务和 MATLAB 与计算子例行程序间的通信。图 3.1 为 MEX 文件的执行流程图。

3.2　C 语言 MEX 文件的编程

在第一节中，我们给出了一个用于处理简单数据的 C 语言 MEX 程序，本节中我们将基于若干个范例程序，具体讲述如何在 C 语言 MEX 文件中对各种类型的 MATLAB 阵列进行处理，使读者对 C 语言 MEX 文件的编程有一个全面的了解。

3.2.1　C 语言 MEX 文件对字符串的操作

1. 范例程序

```
mystringplus.c
/* 程序段 (1) */
#include "mex.h"
#include <string.h>

void mystringplus(char *input_buf0, char *input_buf1, char *output_buf)
{
    strcat(output_buf,input_buf0);
    strcat(output_buf,input_buf1);
}

/* 程序段 (2) */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
```

```
{
    /* 程序段 (3) */
    char *input_buf0, *input_buf1, *output_buf;
    int buflen, buflen0, buflen1, status;

    if(nrhs != 2)
        mexErrMsgTxt("Two inputs required.");
    else if(nlhs > 1)
            mexErrMsgTxt("Too many output arguments.");

    /* 程序段 (4) */
    if ( mxIsChar(prhs[0]) != 1 || mxIsChar(prhs[1]) != 1)
        mexErrMsgTxt("Inputs must be a string.");

    if (mxGetM(prhs[0])! =1 || mxGetM(prhs[1])! =1)
        mexErrMsgTxt("Inputs must be a row vector.");

    /* 程序段 (5) */
    buflen0 = (mxGetM(prhs[0]) * mxGetN(prhs[0]))+1;
    buflen1 = (mxGetM(prhs[1]) * mxGetN(prhs[1]))+1;
    buflen = buflen0 + buflen1-1;

    /* 程序段 (6) */
    input_buf0=mxCalloc(buflen0, sizeof(char));
    input_buf1=mxCalloc(buflen1, sizeof(char));
    output_buf=mxCalloc(buflen, sizeof(char));

    /* 程序段 (7) */
    status = mxGetString(prhs[0], input_buf0, buflen0);
    if(status ! = 0)
        mexWarnMsgTxt("Not enough space. String is truncated.");

    status = mxGetString(prhs[1], input_buf1, buflen1);
    if(status ! = 0)
            mexWarnMsgTxt("Not enough space. String is truncated.");

    /* 程序段 (8) */
    mystringplus(input_buf0, input_buf1, output_buf);

    /* 程序段 (9) */
    plhs[0] = mxCreateString(output_buf);
    return;
}
```

2. 程序注释

程序 mystringplus.c 是一个典型的 C 语言 MEX 源文件，其计算子例行程序和入口

子例行程序存放在一个文件之中。它的功能非常简单，用来将两个字符串合二为一。下面按源程序中的编号对程序代码进行解释：

程序段(1)主要完成了两方面的工作。首先，完成了对头文件 mex.h 和头文件 string.h 的包含，其中 mex.h 已经在前面的章节进行了介绍，这里不再赘述，头文件 string.h 主要包含了一些对字符串操作函数的声明，计算子例行程序中使用的函数 strcat 就在该文件中声明，它是从 C 语言自带的头文件；其次，完成了对计算子例行程序的声明和定义，在计算子例行程序中，连续使用了两次 strcat 函数，从而将两个字符串合二为一。

程序段(2)为入口子例行程序的定义，它是 MATLAB 调用 C 语言 MEX 文件时的入口，是所有 C 语言 MEX 文件所必须具备的内容。

程序段(3)对程序中使用的一些变量进行了定义，并且对用户输入和输出的参数个数进行了检查，程序 mystringplus.c 要求用户必须输入两个参数，并且输出参数的个数不能多于 1，否则程序将报错并退出，函数 mexErrMsgTxt 负责出错信息的输出，并终止程序的运行。

程序段(4)用来对用户输入的参数进行类型和维数检查，程序 mystringplus.c 要求用户输入的所有参数必须为字符串向量，这个功能由函数 mxIsChar 和 mxGetM 来完成。如果类型不匹配，程序将报错并退出，同样由函数 mexErrMsgTxt 负责完成。

程序段(5)的功能是获取两个输入字符串的长度，并计算输出字符串的长度。这里必须注意程序中的加 1 操作，因为在 C 语言中，字符串的总长度为实际的字符个数加上一个标示字符串结束的空字符 NULL。其中函数 mxGetM 和 mxGetN 分别用来获取输入参数的行数和列数。

程序段(6)用来为字符串指针分配内存。这里分配内存时，使用了 MATLAB 提供的库函数 mxCalloc，代替了 C 语言的内存分配函数 calloc，建议读者在进行内存分配时尽量使用 MATLAB 提供的内存分配函数，因为 MATLAB 提供了自动的内存管理功能，它能在 MEX 文件执行结束时，自动地释放所申请的内存，相关的内存分配函数还有 mxMalloc 和 mxRealloc。有关 MATLAB 的内存管理的内容将在后续的章节中以详细介绍。

程序段(7)由两个取数据和两个 if 判断语句构成，用来从输入参数中获取字符串，并判断操作的正确性，取字符串的工作由函数 mxGetString 完成。

程序段(8)为计算子例行程序的调用语句，用来激发计算程序的执行。

程序段(9)用来为输出参数赋值，这个过程通过函数 mxCreateString 来完成。

在以上的程序解释中，只给出所使用函数的大致功能，详细的库函数介绍参见 3.7，3.8 节。

3. 运行结果

对 mystringplus.c 程序进行编译后，可以得到名为 mystringplus.dll 的动态链接库程序，在 MATLAB 命令提示符下键入以下命令：

```
? a = '1234'; b ='5678';
? c = mystringplus(a, b)
```

回车后，可以得到结果

```
c =
        12345678
```

3.2.2 包含多个输出的 C 语言 MEX 文件

对于包含多个输出的C语言MEX文件的编写，基本上同前面单输出的C语言MEX文件的编写方法相同，惟一需要注意的地方是输出参数指针数组 plhs 的指针元素同输出参数之间的对应关系，指针 plhs[0]应指向第一个输出参数，而指针 plhs[1]应指向第二个输出参数，以此类推，则指针 plhs[n]应指向第 n+1 个输出参数。接下来我们对 3.1.1 节中的范例程序作一些简单的改动，使其能够输出两个参数。

首先将程序段(3)中的 if 语句改为如下形式：

```
if(nrhs! =2)
     mexErrMsgTxt("Two inputs required.");
else if(nlhs ! = 2)
     mexErrMsgTxt("Two outputs required.");
```

用以判断输出参数是否为 2，否则报错；

然后，在程序段(9)的后面加上下列语句：

```
plhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);
value = mxGetPr(plhs[1]);
```

其中 value 为一个 double 类型的指针，用语句

```
double *value;
```

在入口子例行程序 mexFunction 的开始进行声明；函数 mxCreateDoubleMatrix 用来创建一个 1×1 的实数矩阵，即一个数量。

对该程序编译后执行可以得到如下结果：

```
? [c, value] = mystringplus(a,b)
c =
    12345678
value =
        0
```

这里 value 的值为零，因为在程序中并未对该值进行任何操作，用户可以自行添加。

3.2.3 C 语言 MEX 文件对 MATLAB 结构体的操作

结构体(Structure)是 MATLAB V5.0 以后的系统提供的一种非常实用的数据类型，与C语言中的结构体极为类似，具体定义可以参见1.2.2节。本小节中将对C语言MEX文件中结构体的操作进行讨论。

1. *范例程序*

C 语言 MEX 文件中对结构体的操作与对字符串的操作相比，要复杂许多，请读者先仔细阅读下面程序 findmax.c 的源代码，在后面将对其进行详细的讲述。

```
/* 程序段 (1) */
#include "mex.h"
#include <string.h>
#define F_NUM_MAX 3 /* 结构体所允许包含的域的最大个数 */
#define S_NUM_MAX 10 /* 结构体对象所包含的元素个数的最大值 */
int max=0;

/* 程序段 (2) */
void findmax( double *grade[], int n)
{
    int i;
    for (i=1; i<n; i++)
    {
        if ( *grade[max]< *grade[i])
            max=i;
    }
    return;
}

/* 程序段 (3) */
void mexFunction( int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[] )
{
    /* 程序段 (4) */
    const char *fnames[F_NUM_MAX];
    const int *dims;
    mxArray *tmp;
    char *name[S_NUM_MAX];
    double *grade[S_NUM_MAX], *p;
    int i,n,numfields,status,buflen;

    /* 程序段 (5) */
    if(nrhs != 1)
        mexErrMsgTxt("One input required.");
    else if(nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");
    else if(! mxIsStruct(prhs[0]))
        mexErrMsgTxt("Input must be a structure.");

    /* 程序段 (6) */
    dims = mxGetDimensions(prhs[0]);
    numfields = mxGetNumberOfFields(prhs[0]);
    n = mxGetNumberOfElements(prhs[0]);

    /* 程序段 (7) */
```

```
    for (i=0; i<numfields;i++)
        fnames[i] = mxGetFieldNameByNumber(prhs[0],i);

    /* 程序段 (8) */
    for (i=0; i<n; i++)
    {
        tmp=mxGetField(prhs[0],i,fnames[0]);
        if (mxIsChar(tmp)! =1)
            mexErrMsgTxt("Element is not type of string");
        buflen = mxGetN(tmp) * mxGetM(tmp)+1;
        name[i] = mxCalloc(buflen, sizeof(char));
        status = mxGetString(tmp,name[i],buflen);
        if(status ! = 0)
            mexWarnMsgTxt("Not enough space. String is truncated.");

        tmp=mxGetField(prhs[0],i,fnames[1]);
        if (mxIsDouble(tmp)! =1)
            mexErrMsgTxt("Element is not type of double");
        grade[i] = mxGetPr(tmp);
    }

    /* 程序段 (9) */
    findmax(grade,n);

    /* 程序段 (10) */
    plhs[0] = mxCreateStructMatrix(1, 1, numfields, fnames);
    tmp = mxCreateString(name[max]);
    mxSetField(plhs[0],0,fnames[0],tmp);
    tmp = mxCreateDoubleMatrix(1,1,mxREAL);
    mxSetPr(tmp, grade[max]);
    mxSetField(plhs[0],0,fnames[1],tmp);

    return;
}
```

2. 程序注释

从程序 findmax.c 的结构上来看，可以清晰地分为两个部分，即计算子例行程序部分和入口子例行程序部分，是一个结构典型的 MEX 文件。该 MEX 文件的输入是一个结构体阵列，而输出为一个 1×1 的结构体矩阵，要求输入的结构体具有两个域，分别为字符串类型和双精度的浮点数据类型，分别代表某学生的名字和该生某一门功课的成绩，程序所完成的功能是从所有的学生中，找出成绩最好的一个并输出其名字和成绩。

向 C 语言的 MEX 文件传送 MATLAB 结构体，基本上与传送其他类型的 MATLAB 数据没有太大的区别，只不过实际得到的数据为 mxArray 结构体类型，这主要是由 MATLAB 结构体数据类型的具体构成决定的，因为 MATLAB 结构体的每一个域仍然

是一个 MATLAB 阵列。所以在 MATLAB 应用程序接口中，相关于 MATLAB 结构体的操作函数的返回值类型均为 mxArray *。

在 C 语言 MEX 文件中对 MATLAB 结构体进行操作时，不能像前面章节中对简单的数值类型的操作那样方便地获取和设置数据，而要通过专门的函数 mxGetField 和函数 mxSetField（函数的具体使用说明参见 3.7，3.8 节）。下面同样按源程序 findmax.c 中的编号对程序进行详细的解释：

程序段(1)主要完成了以下三个任务：第一，对头文件 mex.h 和头文件 string.h 进行了包含；第二，定义了两个宏，用来限制输入结构体的域数和结构体阵列所包含元素的最大个数；第三，定义了一个全局变量，用于计算子例行程序和入口子例行程序进行部分的数据通信。

程序段(2)为计算子例行程序的定义，其完成的功能是从所有的学生中找出成绩最好的那一个。

程序段(3)为入口子例行程序的声明，为固定格式。

程序段(4)为程序中所使用的一些变量的声明，其中声明了三个指针数组，为 fnames，name 和 grade，分别用来存放用户输入结构体的域名字指针、学生的名字指针和成绩指针。

程序段(5)用来对输入和输出参数进行检查，输入参数个数必须为 1 且必须为结构体类型，而输出参数个数不可以大于 1，否则程序将报错并退出。

程序段(6)使用了三个以 mx-为前缀的库函数，分别求得了输入结构体阵列的维数、域数和元素的个数，函数的具体使用说明参见 3.7，3.8 节。

程序段(7)的作用是获得各域的域名，使用了函数 mxGetFieldNameByNumber。

程序段(8)的功能是从输入的结构体阵列中取出数据，并分别存放入指针数组 name 和 grade。

程序段(9)是对计算子例行程序的调用。

程序段(10)为对输出的结构体矩阵进行初始化并赋值，其中使用了 API 的库函数 mxCreateStructMatrix 构造了一个 1×1 的结构体矩阵。

3. 运行结果

对该文件进行编译后，可以得到在 MATLAB 可执行的程序 findmax.dll。在 MATLAB 命令提示符下键入以下命令：

```
? students(1).name = 'Li Hong';
? students (1).grade = 74;
? students (2).name = 'Zhang Yang ';
? students (2).grade = 82;
? students (3).name = 'Wang Wei';
? students (3).grade = 67;
? students (4).name = 'Liu Kai';
? students (4).grade = 92;
? y = findmax(students)
```

回车后可以得到如下结果：

```
y =
name: 'Liu Kai'
     grade: 92
```

3.2.4 C 语言 MEX 文件对 MATLAB 单元阵列的操作

单元阵列(cell array)与 MATLAB 结构体相同，也是 MATLAB V5.0 以后的系统所提供的一种新型的数据类型，它可以包含不同的数据类型的阵列元素，这是其最大的特点，具体的定义参见 1.2.2 小节。本小节中，将针对 C 语言 MEX 文件中对 MATLAB 单元阵列的操作进行讲解。

1. 范例程序

对 MATLAB 单元阵列的操作同样较为复杂，在使用时必须非常小心，请读者先仔细阅读下面的范例程序 uppercase.c，在本节的第二部分，将对它进行详细的解释。

```
/* 程序段 (1) */
#include "mex.h"
#include <string.h>

void uppercase(char *buf)
{
     char *upper;
     upper = strupr(buf);
     buf = upper;
     return;
}

/* 程序段 (2) */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[] )
{
     /* 程序段 (3) */
     int rows, cols, i, j, count, ndims, nindex;
     int buflen, status;
     int subs[2];
     char *buf;
     double *pr, *y;
     mxArray *cell_element_pr, *tmp;

     /* 程序段 (4) */
     if(nrhs! =1)
          mexErrMsgTxt("One input required.");
     else if(nlhs > 1)
          mexErrMsgTxt("Too many output arguments.");
```

```
    else if(! mxIsCell(prhs[0]))
        mexErrMsgTxt("Input must be a cell.");

    ndims = mxGetNumberOfDimensions(prhs[0]);
    if (ndims > 2)
        mexErrMsgTxt("Input must be a cell matrix.");
    /* 程序段 (5) */
    rows = mxGetM(prhs[0]);
    cols = mxGetN(prhs[0]);
    plhs[0] = mxCreateCellMatrix(rows, cols);

    /* 程序段 (6) */
    for (i=0; i < rows; i++)
    {
        for(j=0; j < cols; j++)
        {
            subs[0] = i;
            subs[1] = j;
            nindex = mxCalcSingleSubscript(prhs[0],2,subs);
            cell_element_pr = mxGetCell(prhs[0], nindex);

            if (mxIsChar(cell_element_pr))
            {
                buflen = mxGetM(cell_element_pr) * mxGetN(cell_element_pr) + 1;
                buf = mxCalloc(buflen, sizeof(char));
                status = mxGetString(cell_element_pr, buf, buflen);
                if(status != 0)
                    mexWarnMsgTxt("Not enough space. String is truncated.");

                uppercase(buf);

                tmp = mxCreateString(buf);
                mxSetCell(plhs[0], nindex, tmp);
                mxFree(buf);
            }
            else
                mxSetCell(plhs[0], nindex, mxDuplicateArray(cell_element_pr));
        }
    }
}
```

2. 程序注释

在 MATLAB 应用程序接口函数库中，提供了完善的库函数，包括函数 mxGetCell、函数 mxSetCell、函数 mxCreateCellArray 和函数 mxCreateCellMatrix，用来完成对

MATLAB 单元阵列的访问和创建，使用它们可以方便地对 MATLAB 单元阵列数据对象进行操作。与对 MATLAB 结构体的操作相同，所有从 MATLAB 环境中传向 C 语言环境的 MATLAB 单元阵列数据均为 mxArray 类型，这一点必须非常注意，因为在 MATLAB 中，单元阵列的所有元素同样为一个阵列。

程序 uppercase. c 是一个典型的 C 语言 MEX 程序，主要由计算子例行程序和入口子例行程序组成，其功能是将输入的 MATLAB 单元阵列中的所有字符串类型的元素都转换为大写形式，而对于其他的数据类型原封不动地保留，程序的各部分功能如下：

程序段(1)完成了对一些必须的头文件如 mex. h 和 string. h 的包含，并且定义和声明用于完成字符串大写化的计算子例行程序，该程序通过指针来完成与入口子例行程序的数据交换任务。

程序段(2)为标准的 C 语言 MEX 文件入口子例行程序的声明语句，为固定格式。

程序段(3)对程序中使用的各类型变量进行了声明，其中 mxArray 结构体指针变量 cell _ element _ pr 和 tmp 分别用来存放从输入的单元阵列中获取的数据和向输出的单元阵列写入的数据。

程序段(4)主要用于完成对输入和输出变量的检查，该程序要求必须有一个输入，输出不得多于一个，而且输入参数必须为单元阵列类型，否则程序将报错并退出。

程序段(5)使用函数 mxGetM 和函数 mxGetN 获取输入单元阵列的行数和列数，并以它们为参数使用函数 mxCreateCellMatrix 创建一个单元阵列矩阵。

程序段(6)是程序中最为重要的一个部分。它由一个 for 循环组成，在循环体中，首先使用函数 mxCalcSingleSubscript 和函数 mxGetCell 从输入的单元阵列中取出数据，并存放于 mxArray 类型的指针变量 cell _ element _ pr 中；然后对取出数据的类型进行判断，如果为字符串类型则调用计算子例行程序，全部转换为大写，再利用返回的数据写入用于输出的单元阵列；若数据为非字符串类型，则保留不动，原样写入输出单元阵列，这里注意函数 mxDuplicateArray 的使用，它构建了一个指针变量 cell _ element _ pr 的副本，用于输出，这个步骤非常必须，否则，程序编译通过后执行时，会出现大量的警告信息。

程序中所使用函数的详细使用说明，请参见 3. 7，3. 8 节。

3. 运行结果

对程序 uppercase. c 进行编译得到可执行的 uppercase. dll 文件后，即可在 MATLAB 命令提示符下键入如下的命令：

```
? A(1, 1) = {'asdf'}; A(1, 2) = {5};
? A(2, 1) = {'hjkl'}; A(2, 2) = {-5};
? p = uppercase(A)
```

回车后就可以得到如下结果：

```
p =
    'ASDF'   [ 5]
    'HJKL'   [-5]
```

3. 2. 5 C 语言 MEX 文件对不同位数数据的操作

在 C 语言 MEX 文件中，用户可以创建和操作多种类型的数值型数据，例如有符号的

8 位、16 位和 32 位数据以及无符号的 8 位、16 位和 32 位数据。在 MATLAB 的应用程序接口中，提供了一系列的库函数用以支持对这些数据类型的操作，其中函数 mxCreateNumericArray 用来创建一个指定维数、指定数据类型和指定元素个数的数值阵列，数据类型通过函数的一个 mxClassID 类型输入参数确定，mxClassID 为枚举类型，在头文件 matrix.h 中进行声明，其具体的定义如下：

```
typedef enum
{
    mxCELL_CLASS = 1,
    mxSTRUCT_CLASS,
    mxOBJECT_CLASS,
    mxCHAR_CLASS,
    mxSPARSE_CLASS,
    mxDOUBLE_CLASS,
    mxSINGLE_CLASS,
    mxINT8_CLASS,
    mxUINT8_CLASS,
    mxINT16_CLASS,
    mxUINT16_CLASS,
    mxINT32_CLASS,
    mxUINT32_CLASS,
    mxINT64_CLASS,
    mxUINT64_CLASS,
    mxUNKNOWN_CLASS = -1
} mxClassID;
```

在函数 mxCreateNumericArray 中，可以使用除 mxSTRUCT_CLASS，mxCELL_CLASS 和 mxOBJECT_CLASS 之外的所有值；函数 mxGetImagdata 和 mxGetData 则分别用来获取用户所创建的阵列的虚数数据指针和实数数据指针。用户可以使用 C 语言对这些数据进行数学运算，然后将结果回传给 MATLAB，虽然目前在 MATLAB 环境中，还没有提供任何对以上这些数据类型的数学运算和操作函数，但是 MATLAB 可以正确地接受这些数据，并且显示它们。下面是一个非常简单的操作 16 位无符号数的 MEX 文件源程序，它对所有的数据进行模 5 运算，由于程序较为简单，所以不对程序进行详细的解释。

```
/* 头文件 */
#include <string.h>
#include <math.h>
#include "mex.h"

/* 宏定义 */
#define NDIMS 2
#define TOTAL_ELEMENTS 4

/* 计算子例行程序 */
void mod5(unsigned short *x)
```

```
{
    unsigned short div=5;
    int i, j;
    for(i=0;i<4;i++)
    {
        *(x+i) = *(x+i) % div;
    }
}

/* 入口子例行程序 */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
    /* 变量定义 */
    const int dims[]={2,2};
    unsigned char *start_of_pr;
    unsigned short data[]={56,23,31,47};
    int bytes_to_copy;

    /* 调用计算子例行程序 */
    mod5(data);

    /* 创建一个 2×2 的无符号 16 位整数 */
    plhs[0] = mxCreateNumericArray(NDIMS,dims,
                    mxUINT16_CLASS,mxREAL);

    /* 为创建的矩阵的实数部分赋值 */
    start_of_pr = (unsigned char *)mxGetPr(plhs[0]);
    bytes_to_copy = TOTAL_ELEMENTS * mxGetElementSize(plhs[0]);
    memcpy(start_of_pr,data,bytes_to_copy);
}
```

对该程序编译后直接在 MATLAB 命令提示符下键入文件名就可以得到运行结果：

```
? mod5
ans =
     1     1
     3     2
```

3.2.6　C 语言 MEX 文件对复数的操作

复数阵列是 MATLAB 最基本的处理对象。当在 MATLAB 工作环境中时，对复数阵列的处理非常简单，只需将该阵列视为一个整体，直接进行计算和处理即可；而在 C 语言中，没有提供对复数类型的支持，在对复数进行处理时，必须将复数分为实部和虚部进行分别计算。所以在 MATLAB 应用程序接口中分别为实部和虚部的处理提供了相应的库函数对，包括 mxGetPr 和 mxGetPi，mxSetPr 和 mxSetPi。MATLAB 为用户提供了一个

结构非常清晰的对复数进行操作的 MEX 文件源程序 convec.c，其功能为完成两个复数阵列的卷积运算。该文件存放于目录

MATLAB 根目录\EXTERN\EXAMPLES\REFBOOK

下。下面将程序显示如下，并在程序中通过注释作一些简单的说明。

```
/* 头文件 */
#include "mex.h"

/* 计算子例行程序 */
void convec( double *xr, double *xi, int nx,
             double *yr, double *yi, int ny,
             double *zr, double *zi)
{
    /* 变量的初始化 */
    int i,j;
    zr[0]=0.0;
    zi[0]=0.0;

    /* 复数向量的卷积运算 */
    for(i=0; i<nx; i++)
    {
        for(j=0; j<ny; j++)
        {
            *(zr+i+j) = *(zr+i+j) + *(xr+i) * *(yr+j) - *(xi+i) * *(yi+j);
            *(zi+i+j) = *(zi+i+j) + *(xr+i) * *(yi+j) + *(xi+i) * *(yr+j);
        }
    }
}

/* 入口子例行程序 */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
    double *xr, *xi, *yr, *yi, *zr, *zi;
    int    rows, cols, nx, ny;

    /* 检查输出输入变量的个数 */
    if(nrhs != 2)
      mexErrMsgTxt("Two inputs required.");
    if(nlhs > 1)
      mexErrMsgTxt("Too many output arguments.");

    /* 检查输入是否为向量 */
    if( mxGetM(prhs[0]) != 1 || mxGetM(prhs[1]) != 1 )
      mexErrMsgTxt("Both inputs must be row vectors.");
```

```
    rows = 1;

    /* 确认所有输入为复数类型 */
    if( ! mxIsComplex(prhs[0]) || ! mxIsComplex(prhs[1]) )
      mexErrMsgTxt("Inputs must be complex. \n");

    /* 获得输入变量的元素个数 */
    nx = mxGetN(prhs[0]);
    ny = mxGetN(prhs[1]);

    /* 获得指向输入参数实部和虚部数据的指针 */
    xr = mxGetPr(prhs[0]);
    xi = mxGetPi(prhs[0]);
    yr = mxGetPr(prhs[1]);
    yi = mxGetPi(prhs[1]);

    /* 构造输出矩阵,并获取实部和虚部数据的指针 */
    cols = nx + ny - 1;
    plhs[0] = mxCreateDoubleMatrix(rows, cols, mxCOMPLEX);
    zr = mxGetPr(plhs[0]);
    zi = mxGetPi(plhs[0]);

    /* 调用计算子例行程序 */
    convec(xr, xi, nx, yr, yi, ny, zr, zi);

    return;
}
```

对该程序编译后键入如下命令执行后结果为

```
? x = [3.000 -1.000i, 4.000 + 2.000i, 7.000 -3.000i];
? y = [8.000 -6.000i, 12.000 + 16.000i, 40.000 -42.000i];
? z = convec(x,y)
z =
   1.0e+002 *
   Columns 1 through 4
     0.2400             0 - 0.2600i    0.6200 + 0.6400i    2.0400 - 0.3600i
   Columns 5 through 8
     0.1600 - 1.4400i    1.8400 + 1.9200i    4.1200 - 2.0400i         0 - 4.1400i
   Column 9
   -1.2600
```

3.2.7　C 语言 MEX 文件对稀疏矩阵的操作

稀疏矩阵(sparse array)是 MATLAB 中一种较为特殊的阵列类型,其存储方式同常规阵列存在着较大的差别,详细说明请读者参见 1.2.1 节内容。在 C 语言的 mxArray 结构体的声明中,专门针对稀疏矩阵而作了相应的设计。当判断一个矩阵为稀疏矩阵时,除

了常规的参数 pr 和 pi 外，还使用了另外三个参数，分别为 nnz、ir 和 jc，它们各自的含义请读者参见 2.1.3 节内容。在 MATLAB 的应用程序接口函数库中，为用户提供了一系列的库函数，允许用户在 C 语言 MEX 文件中方便地创建和操作稀疏矩阵，其中函数 mxGetIr、mxGetJc、mxSetIr 和 mxSetJc 就分别用来获取和设置稀疏矩阵参数 ir 和 jc 的值，而 mxCreateSparse 则用来创建一个稀疏矩阵。下面以一个极为简单的例子程序 dbl_sparse.c 来说明如何使用 C 语言 MEX 文件对稀疏矩阵进行操作。在该程序中，首先从用户输入得到一个稀疏矩阵，然后获取其各参数，并对所有的非零元素进行加倍，然后再创建一个新的稀疏矩阵，将计算后的结果赋值给该矩阵并输出，程序较为简单，因此仅在程序中加以注释。

```
/* 头文件包含 */
#include <string.h>
#include "mex.h"

/* 宏定义，用于限定稀疏矩阵的大小及非零元素个数 */
#define NZMAX 4
#define ROWS 4
#define COLS 2

/* 计算子例行程序，用于将输入稀疏矩阵的非零元素加倍 */
void dbl_elem(double elem[], int n)
{
  int i;
  for(i = 0; i < n; i++)
    elem[i] = 2 * elem[i];
  return;
}

/* 入口子例行程序 */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
    /* 定义各种变量 */
    double  *pr_data;
    int     *ir_data;
    int     *jc_data;
    double  *px ;
    int     nz_elem, m, n;

    /* 检查输入输出参数的个数 */
    if (nrhs != 1)
      mexErrMsgTxt("Two inputs required.");
    if (nlhs > 1)
       mexErrMsgTxt("Only one output required.\n");
```

```
    /* 检查输入矩阵是否为稀疏矩阵 */
    if (! mxIsSparse(prhs[0]))
        mexErrMsgTxt("Input must be a sparse matrix.\n");
    /* 检查输入稀疏矩阵的非零元素个数是否超标 */
    nz_elem = mxGetNzmax(prhs[0]);
    if (nz_elem > NZMAX)
        mexErrMsgTxt("The nonzeos elements are too much.\n");

    /* 检查输入稀疏矩阵的行列数是否超标 */
    m = mxGetM(prhs[0]);
    n = mxGetN(prhs[0]);
    if (m > ROWS || n > COLS)
        mexErrMsgTxt("The sparse matrix is too big.\n");

    /* 获取输入稀疏矩阵的各种参数 */
    pr_data = mxGetPr(prhs[0]);
    ir_data = mxGetIr(prhs[0]);
    jc_data = mxGetJc(prhs[0]);

    /* 调用计算子例行程序,加倍非零元素 */
    dbl_elem(pr_data, nz_elem);

    /* 创建一个输出稀疏矩阵,并且命名为"Sparse" */
    plhs[0] = mxCreateSparse(m, n, nz_elem, mxREAL);
    mxSetName(plhs[0],"Sparse");

    /* 为新创建的稀疏矩阵赋值 */
    px=mxGetPr(plhs[0]);
    memcpy((void *)px, (void *)pr_data, nz_elem * sizeof(double));

    /* 为新创建的稀疏矩阵构造行索引 */
    memcpy((void *)mxGetIr(plhs[0]), (void *)ir_data, nz_elem * sizeof(int));

    /* 为新创建的稀疏矩阵构造列索引 */
    memcpy((void *)mxGetJc(plhs[0]), (void *)jc_data, nz_elem * sizeof(int));

    return;
}
```

对该文件编译后,键入如下命令可以得到输出为:

```
? a=[5.9 0;0 5.9;6.2 0;0 6.2]; %创建普通矩阵
? b=sparse(a);                  %转化为稀疏矩阵
? c=dbl_sparse(b)
c =
    (1,1)       23.6000
```

```
(3,1)       24.8000
(2,2)       23.6000
(4,2)       24.8000
```

3.2.8　C 语言 MEX 文件对多维阵列的操作

多维阵列(multidimension array)是 MATLAB V5.0 以后的系统提供的一种新型的数据类型，在本书的 1.2.1 小节中已经对阵中数据类型进行了一定介绍。在 MATLAB 应用程序接口中，为了完成与 C 语言 MEX 文件交换多维阵列数据信息的任务，接口提供了完善和丰富的库函数。下面以例子程序 dbl _ multi.c 来说明在 C 语言 MEX 文件中对多维阵列的创建和操作，该程序从用户输入获得一个三维的多维阵列，从其中获得各种参数，并创建一个同样大小的三维阵列，将所有的阵列元素加倍后，赋给新的阵列。该程序，不同于典型的 C 语言 MEX 文件，它没有计算子例行程序，所有的任务均在入口子例行程序中完成，可以看出计算子例行程序不一定是必须的。但是作者强烈建议严格按格式编写，因为这样使程序的结构清晰，有利于阅读和修改。

```
/* 头文件包含 */
#include "mex.h"

/* 入口子例行程序 */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    /* 变量定义 */
    double *prx, *pry;
    int ndims, i, ele_num;
    const int *dims;

    /* 检查输入输出参数的个数以及输入参数的类型 */
    if ( nrhs != 1 )
        mexErrMsgTxt("One input required!");
    else if ( nlhs > 1 )
        mexErrMsgTxt("Too many output arguments!");
    else if (! mxIsNumeric(prhs[0]))
        mexErrMsgTxt("Input must be numeric!");

    /* 判断输入多维阵列的维数 */
    ndims = mxGetNumberOfDimensions(prhs[0]);
    if (ndims != 3)
        mexErrMsgTxt("Input dimensions must be three!");

    /* 获取多维阵列每一维的大小和总的元素个数 */
    dims = mxGetDimensions(prhs[0]);
    ele_num = mxGetNumberOfElements(prhs[0]);
```

```
    /* 构造新的多维阵列，为非复数的 double 类型 */
    plhs[0] = mxCreateNumericArray(ndims, dims, mxDOUBLE_CLASS, mxREAL);

    /* 获取数据指针 */
    prx = mxGetPr(prhs[0]);
    pry = mxGetPr(plhs[0]);

    /* 完成元素加倍功能 */
    for (i = 0; i < ele_num; i++)
    {
        *(pry + i) = *(prx + i) * 2;
    }
    return;
}
```

程序编译后，在 MATLAB 命令提示符下键入如下命令，可得：

```
? a = randn(3,3,3)
a(:,:,1) =
    -0.4326     0.2877     1.1892
    -1.6656    -1.1465    -0.0376
     0.1253     1.1909     0.3273

a(:,:,2) =
     0.1746    -0.5883     0.1139
    -0.1867     2.1832     1.0668
     0.7258    -0.1364     0.0593

a(:,:,3) =
    -0.0956    -1.3362    -0.6918
    -0.8323     0.7143     0.8580
     0.2944     1.6236     1.2540

? b = dbl-multi(a)
b(:,:,1) =
    -0.8651     0.5754     2.3783
    -3.3312    -2.2929    -0.0753
     0.2507     2.3818     0.6546

b(:,:,2) =
     0.3493    -1.1766     0.2279
    -0.3734     4.3664     2.1335
     1.4516    -0.2728     0.1186

b(:,:,3) =
    -0.1913    -2.6724    -1.3836
```

```
  -1.6647     1.4286     1.7160
   0.5888     3.2471     2.5080
```

randn 函数为产生一个正态分布的随机多维阵列，读者在使用时可能得到与以上不相同的结果。

3.2.9 C 语言 MEX 文件对 MATLAB 函数的调用

通过调用 MATLAB 应用程序接口函数 mexCallMATLAB，用户可以在 C 语言 MEX 文件中调用各种各样的 MATLAB 函数，包括各种 MATLAB 运算符、内建函数以及用户自定义的 MATLAB M 文件，涵盖面非常广泛，为用户提供了极大的方便性。下面通过一个简单的例子来说明如何在C 语言的MEX 文件中调用MATLAB 函数。程序call_matlab.c 的功能是读入一幅图像并显示，该程序也不是典型的C 语言 MEX 文件。

```
/* 头文件包含 */
#include "mex.h"

/* 入口子例行程序 */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
    /* 检查输入输出参数的数量以及输入参数的类型 */
    if ( nrhs != 1 )
        mexErrMsgTxt("One input required!");
    else if ( nlhs > 1 )
        mexErrMsgTxt("Too many output arguments!");
    else if (! mxIsChar(prhs[0]))
        mexErrMsgTxt("Input must be string!");

    /* 调用imread 函数，读入图像 */
    mexCallMATLAB(1, plhs, 1, prhs, "imread");

    /* 调用imshow 函数，用于显示图像 */
    mexCallMATLAB(0, NULL, 1, plhs, "imshow");

    return;
}
```

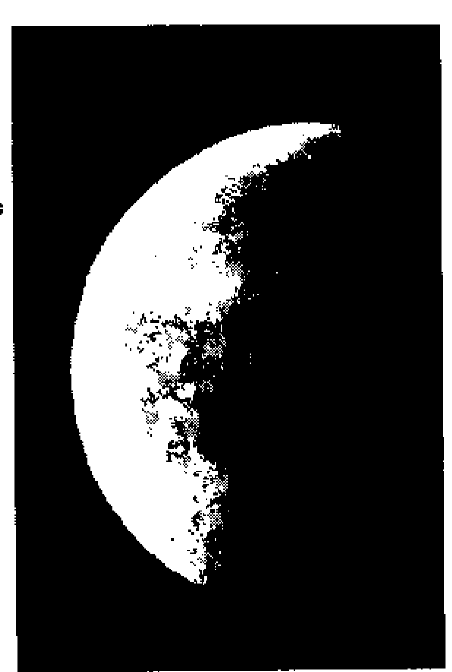


图 3.2 moon.tif

对该文件编译后，键入如下命令执行：

```
? call_matlab('moon.tif');
```

MATLAB 将显示如图 3.2 的图形。

3.3 C 语言 MEX 文件的内存管理

C 语言 MEX 文件的内存管理与常规的C 语言应用程序的内存管理有一定的相似之

处，但是由于 MEX 文件的某些特殊性，例如它并非独立运行，而是在 MATLAB 系统的上下文环境之中执行，所以对于 MEX 文件的内存管理存在着一些特殊的考虑。

3.3.1 自动内存释放

自动内存释放是 MATLAB 系统提供的一种内存管理机制，当一个 MEX 文件执行完毕返回 MATLAB 之后，除了以输出参数列表 plhs[]形式返回给 MATLAB 的计算结果外，任何在 MEX 文件执行过程中创建和使用的临时 mxArray 结构体类型对象均被自动删除，并且任何使用 mxCalloc 函数、mxMalloc 函数和 mxRealloc 函数在 MEX 文件执行过程中分配的内存区域同样将被自动释放，而无须像 C 语言中那样必须显式地释放。

不过一般情况下，MATLAB 建议用户在 MEX 文件中，使用 MATLAB 应用函数库所提供的库函数，包括 mxDestroyArray 和 mxFree，显式地清除文件中所使用的临时 mxArray 对象和动态分配的内存，因为使用这种方法的效率要远远高于依靠自动内存释放机制来清除内存的效率。但是在以下一些特殊情况下，MEX 文件不会正常地结束：

- 当发生调用 API 函数 mexErrMsgTxt 时；
- 当调用 API 函数 mexCallMATLAB 产生错误不能正常执行时；
- 用户使用热键 Ctrl-C 强行终止程序的运行时；
- 当 MEX 文件用完所有的内存时，MATLAB 系统的内存耗尽句柄将立即终止当前 MEX 文件的执行。

若这些情况发生，用户 MEX 文件中的内存管理语句将不会产生任何作用，从而有可能导致内存泄漏的发生。对于一个小心谨慎的程序员来说，在发生第一种和第二种情况时，仍然可以安全地清除所有的临时对象和动态分配的内存；但是当发生第三种和第四种情况时，就无能为力了，这时只有依靠自动内存释放机制来完成内存释放的任务了，从而保证了在任何情况下，都不会发生内存泄漏(Memory Leak)的问题。

可见自动内存释放机制是一种非常有效的内存管理方法。

3.3.2 持久阵列(persistent arrays)

持久阵列是 MATLAB 提供的一种内存变量对象，既可以是一个阵列也可以为一个内存片段，它可以免予被 MATLAB 所提供的自动内存释放机制所释放，通过使用 MATLAB 应用程序接口所提供的库函数 mexMakeArrayPersistent 和 mexMakeMemoryPersistent 可以对持久阵列对象进行定义。然而，持久阵列的创建是一件非常危险的事情，如果在 MEX 文件退出之前，没有将持久阵列对象清除，就会导致内存的泄漏。为了避免这种情况的发生，MEX 文件要求在创建持久阵列对象的同时要使用 API 函数 mexAtExit 注册一个函数，该函数用来完成持久阵列对象的释放工作。例如在以下 MEX 文件中，所定义的持久阵列对象 persistent_array_ptr 通过函数 cleanup 进行了正确的释放。

```
/* 头文件包含 */
#include "mex.h"
/* 全局变量定义 */
static int initialized = 0;
static mxArray *persistent_array_ptr = NULL;
```

```
/* 持久阵列对象清除函数 */
void cleanup(void)
{
     mexPrintf("MEX-file is terminating, destroying array\n");
     mxDestroyArray(persistent_array_ptr);
}

/* 入口子例行程序 */
void mexFunction(int nlhs,mxArray *plhs[],
                 int nrhs,const mxArray *prhs[])
{
     if (! initialized)
     {
          mexPrintf("MEX-file initializing, creating array\n");

          /* 创建持久阵列对象 */
          persistent_array_ptr = mxCreateDoubleMatrix(1, 1, mxREAL);
          mexMakeArrayPersistent(persistent_array_ptr);

          /* 注册 clearup 函数 */
          mexAtExit(cleanup);

          initialized = 1;
          *mxGetPr(persistent_array_ptr) = 1.0;
     }
     else
     {
          mexPrintf("MEX-file executing; value of first array
                    element is %g\n", *mxGetPr(persistent_array_ptr));
     }
}
```

此外同样可以使用 API 函数 mxAtExit 来释放其他一些资源。

3.3.3 复合阵列

在 MATLAB 应用程序接口函数库中，包含了大量的可以直接将内存段赋值给 mxArray 结构体对象的函数，如 mxSetPr，mxSetData，mxSetPi，mxSetCell 等。对于这些内存段的清除工作，MATLAB 应用程序接口函数库提供了相应的函数 mxDestroyArray，该函数在清除整个 mxArray 机构体对象的同时将所有分配的内存段清除。但是这并不是非常可靠，因为对于某些阵列来说，并不能用函数 mxDestroyArray 来清除，复合阵列(hybrid array)就是这样一种阵列，在这种阵列的内存中，既存在可清除的内存段，同时也存在不可清除的内存段，例如 C 语言中的自动变量(automatic variable)就是一种典型的不可清除的内存段，它既不可以用 MATLAB API 的库函数 mxFree 进行清除，也不可以用标准的 C 语言函数 free( )来进行清除。例如下面的代码

```
mxArray *pArray = mxCreateDoubleMatrix(0, 0, mxREAL);
double data[10];
mxSetPr(pArray, data);
mxSetM(pArray, 1);
mxSetN(pArray, 10);
```

定义了一个名为 pArray 大小为 1×10 的复合阵列，其内容是一个普通的 double 类型的自动变量数组。另外一种典型的复合阵列是具有一个或多个只读的阵列域的单元阵列和结构体阵列，这里只读阵列是指那种具有常量限定符的阵列，例如 MEX 文件的输入阵列就是一种只读阵列。

正是由于复合阵列不能被清除，所以它们也不能被前面所讲述的内存自动清除机制所清除，因此不正确地使用复合阵列有可能导致 MEX 文件的崩溃，所以建议读者尽量避免使用复合阵列，虽然它的使用较为方便。

3.4　C 语言 MEX 文件的建立

在 3.2 和 3.3 节中，我们分别讲述了基于 C 语言的 MEX 文件的编写和 C 语言 MEX 文件的内存管理。在本节中，我们将对 C 语言 MEX 文件的建立进行讲述，使读者全面掌握 C 语言 MEX 文件的使用。

3.4.1　C 语言 MEX 文件的建立

在第二章第一节对 MEX 文件的简单介绍中，提供了一种极为简单的 MEX 文件的建立方法，即直接使用 mex 命令，并在其后加上所希望编译的 MEX 文件的源文件名，具体格式如下：

```
mex <MEX 文件名>
```

其使用条件是已经使用命令

```
mex-setup
```

对默认的选项文件进行了正确的配置。对于一般的应用来说，这已经足够了，但是如果用户希望进行一些高级的操作，例如创建一个新的选项文件用于支持 MATLAB 系统没有直接予以提供支持的编译器，或者对编译过程施加更多的控制，这时使用以上格式就无能为力了。

然而，mex 命令是一个功能非常强大的工具，以上命令格式仅仅是其最简单的一种应用方式，它拥有大量的命令控制参数，通过设置这些参数，允许用户使用不同的手段来完成不同的任务，例如通过使用参数-f，用户可以选择指定的选项文件对源程序进行编译，通过使用参数-g 可以让生成的 MEX 文件包含调试信息，以便进行调试，详细的参数列表以及各参数功能见表 3.1，在 MATLAB 命令提示符下也可以通过键入命令

```
mex -h
```

来获取帮助。

表 3.1 mex 命令控制参数表

控制参数名	功 能
-argcheck	用于 MATLAB 应用程序接口函数库库函数的参数检查(仅用于 C 语言 MEX 文件)
-c	用于通知编译器仅对源文件进行编译而不链接
-D<name>[=<def>]	该控制参数仅限于 UNIX 和 Macintosh 平台使用,用于定义 C 预处理程序中的宏<name>,并使其值为<def>
-D<name>	该参数仅限于 Windows 平台使用,用于定义 C 预处理程序的宏<name>
-f <file>	当该参数使用于 UNIX 和 Windows 平台时,其含义为使用<file>所确定的文件为选项文件,当该选项文件不在系统默认的路径和当前路径上时,必须明确地指出选项文件所在的位置,否则系统将报告找不到该选项文件,并退出编译过程。在 Windows 平台上,当对 mex 命令进行 setup 后,就可以忽略该参数的使用,直接对 MEX 文件进行编译。 当该参数使用于 Macintosh 平台时,如果指定了文件<file>,那么系统将该文件作为选项文件;如果没有指定文件<file>,那么系统将使用当前目录中的文件 mexopts 作为默认的选项文件;如果既没有指定文件<file>,在当前目录中也没有文件 mexopts,那么系统将在路径<MATLAB 根目录>\extern\scripts\寻找一个名为 mexopts 的选项文件,如果该文件仍然不存在,则系统报错
-F <file>	当使用于 UNIX 平台时,参数含义为使用文件<file>作为选项文件,并且按次序搜索以下目录,并且使用最先匹配的一个文件: • \<filename> • $HOME\matlab\<filename> • $TMW_ROOT\bin\<filename> 当使用于 Windows 平台时,参数含义为使用文件<file>作为选项文件,目录的搜索次序为: • 当前目录 • mex.bat 文件所在的路径
-g	建立一个包含 debug 信息的可执行的 MEX 文件
-h	列出帮助信息
-I<pathname>	功能为将路径<pathname>包含入编译器的搜索路径
-l <file>	功能为将 MEX 文件与库<file>链接(仅用于 UNIX 平台)
-L <pathname>	功能为将路径<pathname>包含入库函数的搜索路径(仅用于 UNIX 平台)
<name>=<def>	使用于 UNIX 和 Macintosh 平台,用于重载选项文件的变量名设置<name>
-n	非执行标志
-output <name>	创建一个名为<name>的可执行文件,其后缀名由系统自动添加
-O	建立一个优化过的可执行文件
-setup	用于设置默认的选项文件
-U <name>	使用于 UNIX 和 Windows 平台,用于取消 C 预处理程序的宏定义<name>
-V4	建立与 MATLAB V4 版本兼容的 MEX 文件
-v	打印所有的编译器和链接器的设置

对于使用过 C 语言或 FORTRAN 语言以及其他一些高级计算机语言的读者来说,可能已经知道,建立一个源程序的可执行文件一般来说,可以分为两步,即编译(compile)过程和链接(link)过程。对于 MEX 文件的建立来说,同样可以也分为编译和链接两步,在编译过程将对用户输入的源程序进行语法和语义的转换,判断语句的正确性,并生成目标文件;链接过程则是对编译过程生成的目标文件进行进一步的处理,对程序中所使用的静

态或动态链接库函数进行链接，并生成可执行程序。对于 Windows 平台上的编译过程，略有不同，它在编译过程和链接过程的中间增添了一些步骤，称之为预链接(pre-link)过程，用来完成链接过程前的一些准备工作。

通过表 3.1 中的参数，用户可以对 MEX 文件的建立过程进行一定的控制，但是如果用户希望完全地、自定义地控制整个 MEX 文件的建立过程，就必须对所使用的选项文件进行修改了。选项文件在第二章中已经进行了一定的介绍，它包含了针对用户系统上某种编译器的编译、预链接和链接的选项信息，由一系列的变量组成，其中每一个变量代表了 MEX 文件建立过程中的一个步骤。

对于默认选项文件的存放位置，不同的操作系统各不相同。在 UNIX 操作系统中，该选项文件存放于目录<MATLAB 根目录>\bin 中；在 Windows 操作系统中，该选项文件存放于目录<MATLAB 根目录>\bin 中；而在 Macintosh 系统中，该选项文件存放于目录<MATLAB 根目录>\EXTERN\SCRIPTS\FOLDER 中。

此外对于选项文件的定位，不同的操作系统 mex 命令搜索的顺序也各不同。在UNIX操作系统中，命令将首先在当前所在的目录中搜索，默认文件名为 mexopts.sh，然后在用户目录 $HOME\matlab 中进行搜索，最后搜索目录<matlab>\bin；在 Macintosh 操作系统中，默认的选项文件名为 mexopts，mex 命令执行时，首先在当前文件夹中进行搜索，然后搜索路径<MATLAB 根目录>\EXTERN\SCRIPTS\FOLDER。对于任何系统，均可以使用 mex 命令参数-f 来指定一个选项文件，用于 MEX 文件的建立。

3.4.2 基于 Windows 操作系统的 C 语言 MEX 文件的建立流程

在上一小节中，我们对 C 语言 MEX 文件的建立进行了总体上的描述，在本小节中，我们将对基于 Windows 操作系统的 C 语言 MEX 文件的建立进行详细的讲述。

总的说来，基于 Windows 操作系统的 C 语言 MEX 文件的建立流程可以分为三个步骤，即编译(compiling)、预链接(prelinking)和链接(linking)。

1. 编译(compiling)

在 Windows 操作系统中，整个编译过程必须包含以下步骤：

首先，设置编译 MEX 文件所使用的一些环境变量，包括路径变量 PATH，头文件包含变量 INCLUDE 和库文件变量 LIB，例如当用户使用的为 Microsoft Visual C++系统时，PATH、INCLUDE 和 LIB 则分别为 Microsoft Visual C++的安装路径及其所包含的头文件和库文件所在的路径。当用户在自己的操作系统或编译器中已经对这些环境变量进行了注册，例如在 Windows 9x 操作系统中，用户已经在 autoexec.bat 中写入这些变量，或者在 Windows NT 操作系统中，在控制面板中的系统选项中的环境变量属性页中定义了这些变量，那么这时用户就可以将选项文件中的这部分内容注释掉，而不会发生问题。如果这些变量设置不正确，在编译 MEX 文件时，系统将报错，说某些文件找不到，导致编译过程的失败退出；

其次，定义所使用的编译器的名字 COMPILER，在使用 Microsoft Visual C++系统时，COMPILER 应设置为 cl(注：cl 为 Microsoft Visual C++的编译执行文件)；

再次，对编译器的编译标志 COMPFLAGS 进行如下设置：

- 建议使用参数-c，以告诉编译器，只对源文件进行编译而不用链接；
- 使用参数-Zp8，按 8 个字节对准；
- 必须使用参数-D 定义预处理程序宏 MATLAB _ MAX _ FILE；
- 设置编译器优化参数；
- 设置调试信息参数；
- 自由设置其他一些编译器参数。

2. 预链接

预链接过程主要用来动态创建输入函数库，这些函数库包含了 MATLAB 应用程序接口的库函数，以便在MEX 文件中使用。所有的MEX 文件仅仅与MATLAB 发生链接，与之相关的 .DEF 文件放置于目录

<MATLAB 根目录>\EXTERN\INCLUDE

中。对于 MATLAB MAT 文件和引擎文件不太相同，在后面章节中将分别讲述。

3. 链接

整个 MEX 文件的建立过程的最后一个步骤为链接过程，整个过程中必须完成以下操作：

首先，必须定义环境变量 LINKER，用于指明所使用的链接器，在使用 VisualC＋＋时，其定义为：

LINKER ＝ link

其次，必须对链接器进行参数设置，即定义环境变量 LINKFLAGS：

- 使用参数 dll，表明用于创建动态链接库文件；
- 将输出点指向函数 mexFunction；
- 链接在预链接过程产生的输入函数库；
- 自由设置其他一些相关链接器参数。

再次，定义链接优化参数 LINKEROPTIMFLAGS；

第四，定义链接调试信息参数 LINKERDEBUGFLAGS；

第五，在需要的情况下，在环境变量 LINK _ FILE 和 LINK _ LIB 中对链接文件(link-file)标示符和链接库(link-library)进行定义，例如在使用 Watcom C 时，Watcom 使用它们来确定输入的名字是一个变量还是一个命令是一个库；在使用 VisualC＋＋时，这些环境变量可以不用设置；

第六，在环境变量 NAME _ OUTPUT 中使用参数 output 建立一个输出标示符和名字，如下：

NAME _ OUTPUT＝/out："%OUTDIR%%MEX _ NAME%.dll"

其中环境变量 MEX _ NAME 包含了命令行中第一个程序的名字，该环境变量在使用参数 output 时必须被设置，以使 output 正常工作。如果该环境变量没有被设置，编译器默认地使用命令行中的第一个程序名。

4. 对选项文件 msvc60opts.bat 的分析

msvc60opts.bat 是 MATLAB 为 Microsoft Visual C++ 6.0 提供的一个专门用于编译 MEX 文件的选项文件，位于目录

<MATLAB 根目录>\bin

中，其源代码如下，请读者先详细阅读该程序，在后面将对该选项文件进行全面的分析，以加深读者对 C 语言 MEX 文件编译过程的理解。

```
@echo off
rem MSVC60OPTS.BAT
rem ********************************************************************
rem (1) 普通参数设置
rem ********************************************************************

set MATLAB = %MATLAB%
set MSVCDir = %MSVCDir%
set MSDevDir = %MSVCDir%\..\Common\msdev98
set PATH = %MSVCDir%\BIN;%MSDevDir%\bin;%PATH%
set INCLUDE = %MSVCDir%\INCLUDE;%MSVCDir%\MFC\INCLUDE;
%MSVCDir%\ATL\INCLUDE;%INCLUDE%
set LIB=%MSVCDir%\LIB;%MSVCDir%\MFC\LIB;%LIB%

rem ********************************************************************
rem (2) 编译器参数设置
rem ********************************************************************

set COMPILER = cl
set COMPFLAGS = -c -Zp8 -G5 -W3 -DMATLAB_MEX_FILE -nologo
set OPTIMFLAGS = -O2 -Oy-
set DEBUGFLAGS = -Zi
set NAME_OBJECT = /Fo

rem ********************************************************************
rem (3) 库创建命令(预编译过程)
rem ********************************************************************

set PRELINK_CMDS1=lib /def:"%MATLAB%\extern\include\matlab.def"
/machine:ix86 /OUT:%LIB_NAME%1.lib
set PRELINK_CMDS2=lib /def:"%MATLAB%\extern\include\libmatlbmx.def"
/machine:ix86 /OUT:%LIB_NAME%2.lib
set PRELINK_DLLS = lib /def:"%MATLAB%\extern\include\%DLL_NAME%.def"
/machine:ix86 /OUT:%DLL_NAME%.lib

rem ********************************************************************
```

```
rem (4) 链接器参数设置
rem ********************************************************************

set LINKER = link
set LINKFLAGS = /dll /export:mexFunction /MAP %LIB_NAME%1.lib
%LIB_NAME%2.lib /implib:%LIB_NAME%.lib
set LINKOPTIMFLAGS=
set LINKDEBUGFLAGS=/debug
set LINK_FILE=
set LINK_LIB=
set NAME_OUTPUT=/out:"%OUTDIR%%MEX_NAME%.dll"
set RSP_FILE_INDICATOR=@

rem ********************************************************************
rem (5) 资源编译器参数设置
rem ********************************************************************

set RC_COMPILER=rc /fo "%OUTDIR%mexversion.res"
set RC_LINKER=

set POSTLINK_CMDS=del "%OUTDIR%%MEX_NAME%.map"
```

由上面的源代码可以看出，选项文件 msvc60opts.bat 的结构非常清晰，总共可以分为五个部分：

第一部分为普通参数设置，定义了一些关于 MATLAB 以及 C 语言编译器的路径信息，其中%MATLAB%和%MSVCDir%分别代表了 MATLAB 和 Microsoft Visual C++ 6.0 所在的安装路径，其余的一些相关定义，都是建立在它们的基础之上；

第二部分为编译器参数设置，在该选项文件中，通过环境变量 COMPILER 设置了编译器为 cl，并且通过环境变量 COMPFLAGS、OPTIMFLAGS、和 DEBUGFLAGS 对编译器进行了全面的设置，其中一些参数在本小节的第一部分已经进行了介绍，其余的一些含义如下：

```
-G5    代表针对奔腾处理器进行优化处理
-W3    代表设置警告级别为 3 级
-Zi    代表使能调试信息
-O2    代表以最大速度优化
-nologo    代表禁止版权信息
```

第三部分为库创建命令，即预编译过程，用于创建输入函数库；

第四部分为链接参数设置，在这部分内容中，首先通过环境变量 LINKER 设置了链接器的名字为 link，然后通过环境变量 LINKFLAGS 设置了链接器的一些参数选项，如设置了链接为动态链接库文件，使出口点为 mexFunction；同时对于环境变量 LINK_FILE 和 LINK_LIB 进行设置，在使用 VisualC 编译器时，它们可以不用设置，而在使用 Watcom 编译器时则必须设置；设置环境变量 LINKDEBUGFLAGS 为/debug 表示包含调试信息；

第五部分为资源编译器参数设置，这部分内容将在后面介绍。

3.4.3 链接多个文件

在建立C语言MEX文件时，不但可以将若干个目标文件结合在一起构成MEX文件，而且可以同时包含目标库文件，例如在Windows操作系统中，在MATLAB命令提示符下键入如下命令

```
? mex title.c text.obj name.obj thesis.lib
```

将产生一个名为title.dll的动态链接库文件，可在MATLAB工作环境中直接执行。这里必须注意：第一，mex命令可以操作若干种文件格式，包括.c、.obj和lib；第二，在链接多个文件时，生成的MEX文件的名字为文件列表中的第一个文件的名字；第三，在将文件列表时，必须写出文件的扩展名，并且用空格分隔。

mex命令提供的这项功能对于软件开发者来说是非常有用的，它允许用户使用像MAKE那样的开发工具来管理包含多程序源文件的MEX文件项目，仅仅只需要用户编写一个简单的MAKEFILE文件，在这个文件中包含了将用户MEX文件项目中各源文件编译为目标文件的规则，然后就可以激活mex命令将各目标文件结合在一起，从而生成可执行的MEX文件。通过这种方法，用户在建立MEX文件时，不用每次都对源文件进行编译，而只需链接即可。

3.4.4 将C语言MEX文件与动态链接库DLLs链接

将一个动态链接库文件链接到MEX文件中，必须完成两方面的工作，首先，必须在命令行中列出动态链接库的文件名；其次，必须正确地设置选项文件中的环境变量PRELINK_DLLS，其功能主要是用来动态地从用户输入的动态链接库创建一个输入函数库，以便动态链接库可以被链接到MEX文件上。该环境变量包含了产生输入函数库的命令和参数选项，同时，使用变量DLL_NAME包含了命令行中提供的动态链接库的文件名，在选项文件msvc60opts.bat中定义如下：

```
set PRELINK_DLLS=lib/def:"%MATLAB%\extern\include\%DLL_NAME%.def"
/machine:ix86/OUT:%DLL_NAME%.lib
```

3.4.5 C语言MEX文件的版本信息

mex命令可以在建立用户的MEX文件时嵌入一个资源文件，该资源文件包含了版本和其他一些基本信息。在Windows平台上，该资源文件名为mexversion.rc，存放在目录

<MATLAB根目录>\EXTERN\INCLUDE

中。为了在建立MEX文件时嵌入这些信息，必须对选项文件进行一定的设置，即上文中选项文件msvc60opts.bat中的第五部分，它包括了对环境变量RC_COMPILER和环境变量RC_LINKER的设置工作，提供了资源编译器和链接器命令。当对编译器命令进行了设置之后，编译后的资源文件将使用标准的链接命令链接到MEX文件中；如果同时在选项文件中设置了链接命令，那么资源文件将使用这个命令链接到MEX文件中。在选项

文件 msvc60opts.bat 中，只设置了 RC _ COMPILER。

3.5　C 语言 MEX 文件的调试

对于使用 C 语言等高级计算机语言进行过程序设计的读者来说，可能已经深切地体会到，程序的调试对于一个应用程序的建立来说，具有何等重要的意义。因为对于任何一个程序员来说，一次性地将程序设计完成，并且不发生任何错误，几乎是不可能的事情，往往可能发生一些意想不到的问题，并且这些问题发生在程序的运行过程中，非编译过程和链接过程所能够检测到。而程序的调试却能够帮助程序员在程序的运行过程中找到这些隐含的逻辑错误，并加以改进，是程序编制过程中一个非常重要的步骤。正因为如此，MATLAB 也对 MEX 文件提供了全面的调试功能，包括设置断点、单步运行、变量检查等，而且几乎适用于所有的操作系统。

在本节中，将对 Windows 操作系统和 UNIX 操作系统中 C 语言 MEX 文件的调试进行全面地讲述。

3.5.1　Windows 操作系统中 C 语言 MEX 文件的调试

如果读者希望对一个 MEX 文件进行调试，那么必须在建立 MEX 文件时，使用 mex 命令参数-g，格式如下：

```
mex -g MEXfile _ name.c
```

通过该参数告诉编译器，在建立 MEX 文件时，包含所需要的调试信息。若对一个没有包含调试信息的 MEX 文件进行调试，调试器将报告没有调试信息，并且无法正确设置断点，不能完成调试任务。因此，强烈建议读者在首次编译自己的应用程序时，使用参数 -g，以包含调试信息，便于在程序出错时进行调试。当确认程序完全无误后，再重新进行编译，生成不包含调试信息的可执行程序，这样可提高程序的运行速度和减小执行程序的代码长度。

1. 使用 Microsoft Visual C++ 6.0 集成环境进行调试

使用 Microsoft Visual C++ 6.0 集成环境进行 C 语言 MEX 文件的调试时，必须按如下步骤操作：

第一步，进入 Visual C++ 6.0 的集成编译环境，点取 Open 菜单项，选择希望调试的 MEX 文件，这时 Visual C++ 6.0 将会为用户构造一个工程；

第二步，再次点取 Open 菜单项，打开 MEX 文件的源程序；

第三步，在 Visual C++ 6.0 所生成的项目空间中，选择下拉式菜单 Project 中的菜单项 Settings，将弹出一个对话框，见图 3.3。点取其中的 Debug 属性页，在名为 Executable for debug session 的编辑框中输入 MATLAB 的可执行文件，必须包含全路径名，其他所有的编辑框都保留为空；

第四步，选择下拉式菜单 Build，选择其中的 Debug 菜单项，并且点取 Go 子菜单项，开始整个调试过程；

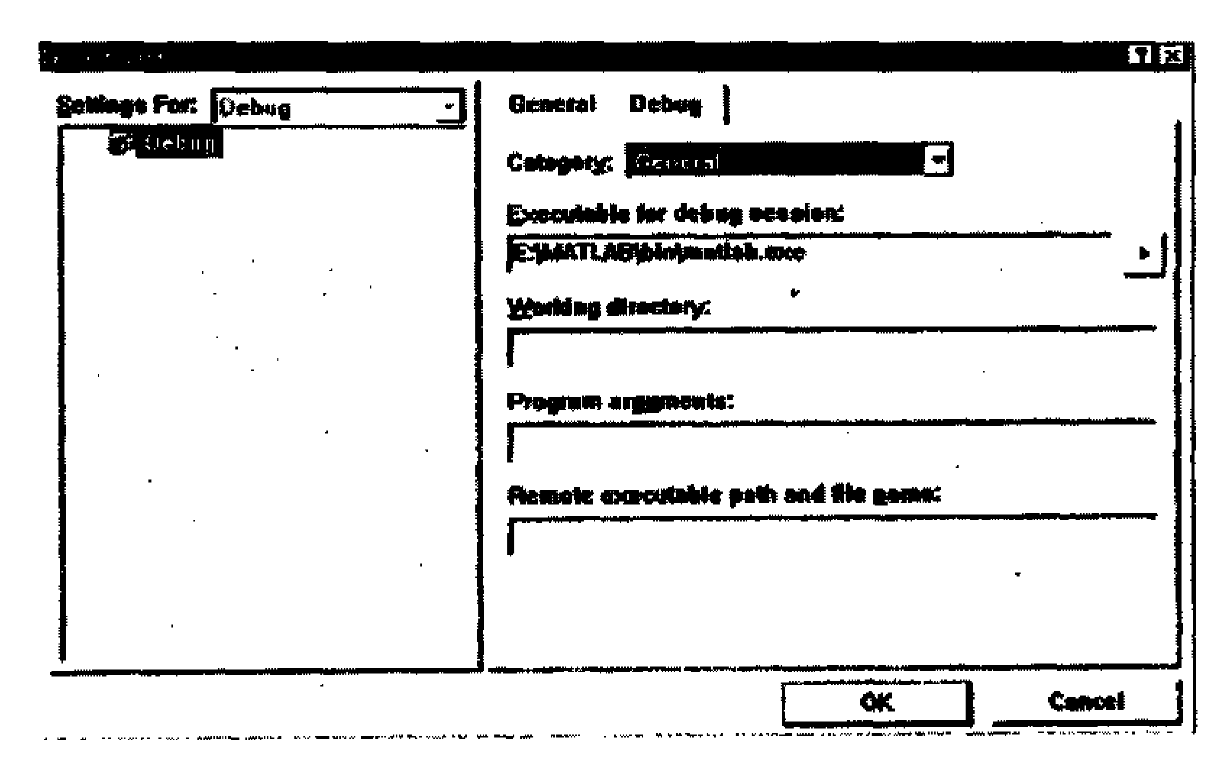


图 3.3　下拉式菜单 Project 中菜单项 Settings 的对话框

第五步，调试过程开始后，Visual C++ 6.0 调试器将调用 MATLAB，这时将首先弹出一个对话框，如图 3.4 所示：

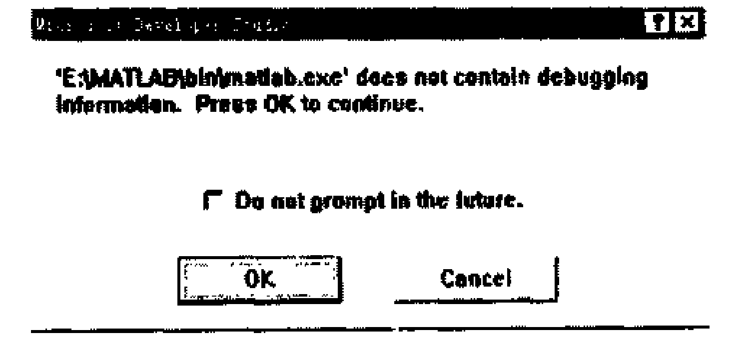


图 3.4　Microsoft Deveploper Studion 警告对话框

用以提示用户，可执行文件 matlab.exe 没有包含调试信息，这是正常现象，用户可以忽略这个警告，并选取复选框，点取 OK 按钮继续；然后 VisualC++将产生一个新的 MATLAB 进程，并且将当前的路径设置到所需要调试的 MEX 文件的路径。

这时就可以在 MATLAB 环境下键入 MEX 文件名和参数，在 VisualC++集成环境中设置断点，进行调试了。具体如何设置断点，如何单步运行，如何观察变量的内容，请读者自行参阅相关的 Visual C++集成环境的帮助。

2. 使用 Watcom C 进行调试

使用 Watcom C 调试器对 MEX 文件进行调试时，必须完成以下步骤：

首先，启动 Watcom C 调试器，无论是在 DOS 命令框下键入命令

WDW

或是直接在 Windows 下运行，这时调试器将开启一个新的程序窗口(New Program)，选择 Cancel；

其次，从 Break 菜单中选择菜单项 On Image Load，并且用大写字母键入希望调试的 MEX 文件名，然后选择 ADD 并且点击 OK 关闭窗口；

第二，打开 MEX 文件的源文件；

第三，从 File 菜单项中选择菜单项 Open，键入包括全路径名的 MATLAB 的可执行文件的位置，并选择 OK，例如在作者的系统上为：

```
e:\matalb\bin\matlab.exe
```

第四，开始调试后，Watcom C 同样会调用 MATLAB，并开启一个 MATLAB 的窗口，当在源文件中设置断点后，用户就可以在 MATLAB 的工作环境中键入 MEX 文件名和参数对 MEX 文件进行调试了；不过 Watcom C 调试器将会报出类似如下的警告信息：

```
LDR: Automatic DLL Relocation in matlab.exe
LDR: DLL filename.dll base <number> relocated due to collision
   with matlab.exe
```

用户可以选择 OK，忽略这些信息，而不会对调试过程产生任何影响。

3.5.2　UNIX 操作系统中 C 语言 MEX 文件的调试

在 UNIX 操作系统中，对 C 语言 MEX 文件的调试与在 Windows 操作系统中相比存在较大的差异，不过相同的是，对希望调试的 MEX 文件必须使用 mex 命令参数-g 进行编译，否则无法进行调试。

希望对 MEX 文件进行调试，首先必须在一个调试器内启动 MATLAB 的工作环境，则可以通过如下命令行办到：

```
matlab -Ddbx
```

其中 matlab 为 MATLAB 的可执行文件名，而 dbx 为一个在 UNIX 环境下的调试工具，参数-D 则告诉 MATLAB 在调试器的空间内运行。当调试器将 MATLAB 调入内存之后，就可以使用命令 run 启动它，并在 MATLAB 工作环境内键入命令

```
dbmex on
```

使能对 MEX 文件进行调试。然后用户必须告诉调试器 MEX 文件所在的具体位置及名字，并且列出 MEX 文件的源代码，设置断点后，就可以像执行普通的 MEX 文件一样，在 MATLAB 环境中，开始对 MEX 文件调试了。

3.6　Microsoft Visual C++集成环境中 MEX 文件的建立

如果读者按照前面讲述的内容对一些例子文件进行了编译和调试，将会发现这种开发的方法非常的麻烦，尤其是习惯了使用各种集成开发环境如 Microsoft Visual C++ 6.0 以及 Borland C++ 5.2 的读者，更会觉得极不适应。建立一个 MEX 文件，首先必须在某个编辑器中进行源代码的编写，然后存盘后回到 MATLAB 的工作环境中，进行编

译，若发现错误，则必须回到原来的文件编辑器，按照 MATLAB 的错误提示，逐行地对源代码进行查错修改，之后再回到 MATLAB 的工作环境中进行编译，如此往复，直至程序没有错误为止，一个 MEX 文件才宣告完成。整个过程需要来回地在文件编辑器和 MATLAB 工作环境之间切换，非常不方便，而且过程中还没有加入调试步骤，否则将更加麻烦。

在本节中，我们将讲述一种在 Microsoft Visual C++集成环境中建立 MEX 文件并调试的方法。

3.6.1 Microsoft Visual C++集成环境中建立 MEX 文件的步骤

在 Microsoft Visual C++集成环境中建立 MEX 文件，可以分为 9 个步骤，如下：

第一步，进入 Microsoft Visual C++ 6.0 集成环境，选择 File 菜单中的 New 菜单项，将弹出一个如图 3.5 所示的对话框，选择其中的"win32 Dynamic-link Library"项，并且按要求输入文件名，创建一个空的 win32 的动态链接库的工程；

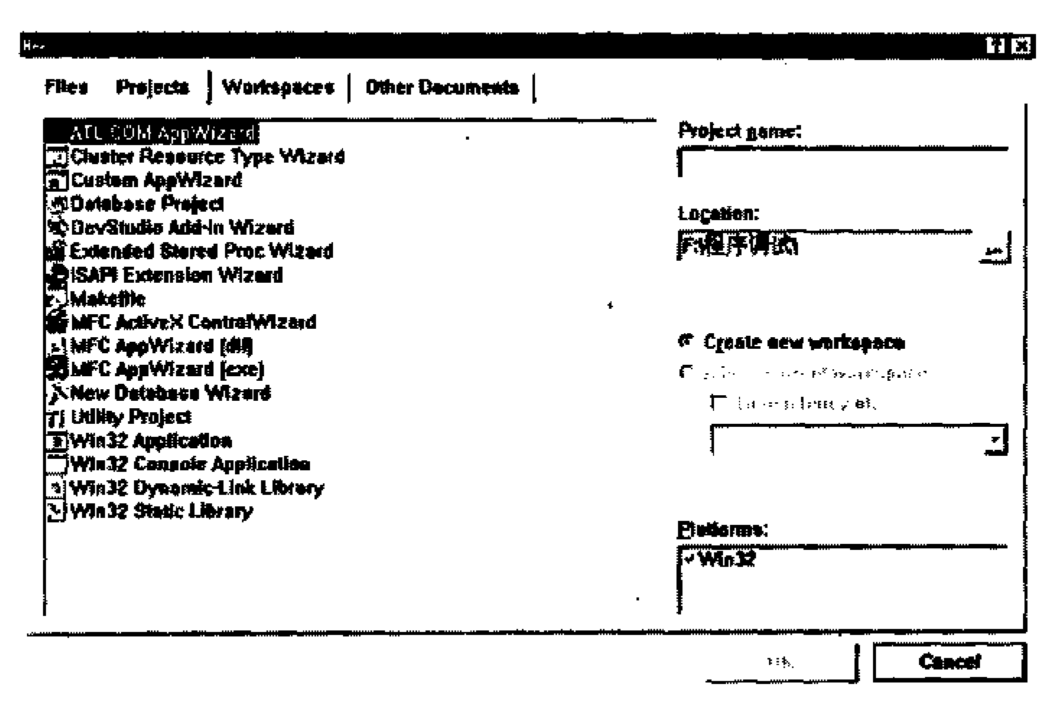


图 3.5　VisualC++6.0 File New 对话框

第二步，点取下拉式菜单 Project 中的 Add to Project 菜单项，然后选择其中的 Files 选项，添加一个与用户创建的工程同名的定义文件，其后缀为 .def，为了说明方便，这里假定工程名为 mexfunc，则定义文件名为 mexfunc.def；

第三步，在 mexfunc.def 中键入以下两行代码：

```
LIBRARY mexfunc.DLL
EXPORTS mexFunction
```

这是 win32 动态链接库编写的典型格式，其中第一行代码表示产生的动态链接库的名字为 mexfunc.DLL，第二行代码表示输出函数名为 mexFunction；这里除了第一行中的文件名可以随用户的不同工程而不同之外，其他任何部分均不能有丝毫改动；

第四步，创建用户自己的 C 语言 MEX 文件，例如

```
/* mexfunc.c */

#include "mex.h"
void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[] )
{
    printf("Edzko Smid\n");
    return;
}
```

第五步，在 DOS 命令框下使用 LIB 命令，根据定义文件 MATLAB.def 创建一个 MEX 函数的输入库，例如：

```
C:\>LIB /DEF:C:\MATLAB\EXTERN\INCLUDE\MATLAB.DEF /MACHINE:IX86
/OUT:mymeximports.lib
```

不过在执行此命令之前，必须确保系统中已经设置了以下路径：

```
C:\> SET PATH = C:\PROGRA~1\DEVSTU~1\VisualC\BIN;
C:\PROGRA~1\DEVSTUD~1\SHARED~1\BIN;
```

一旦库文件 mymeximports.lib 被创建之后，用户就可以在任何一个 MEX 文件中使用该库文件，而不用重复生成，其中定义文件 MATLAB.def 存放于目录

<MATLAB 根目录>\extern\include

中；

第六步，将上一步生成的库文件 mymeximports.lib 输入到当前工程中，方法同步骤 2；

第七步，将其他一些必须的函数或库文件添加到当前工程中，方法同步骤 2；

第八步，点取下拉式菜单 Project 中的 Settings 菜单项，将弹出一个如图 3.6 所示的对话框，选取其中的 Link 属性页，在 Category 下拉框中选取“Input”，然后在Additional library path 编辑框中输入“c:\matlab\extern\lib”，以指明额外的库路径；

第九步，同样如图 3.7 所示，在 Settings 菜单项弹出的对话框中，选取 C/C++属性页，在 Category 下拉框中选取“Preprocessor”，然后在Additional include directories 编辑框中输入“c:\matlab\extern\include”，以指明额外的头文件的路径，并且在Preprocessor definitions 编辑框中输入宏 MATLAB_MEX_FILE；

当完成以上九步工作之后，就可以对 MEX 源程序进行编译和链接，生成可执行的 MEX 文件了。

3.6.2 Microsoft Visual C++集成环境中 MEX 文件的调试

在 Microsoft Visual C++集成环境中对 MEX 文件调试，相对于 MEX 文件的建立来说就简单许多了，因为在 MEX 文件的建立过程中已经完成了大部分的工作，提供调试功能只需要非常简单的一步设置，即在 Settings 弹出的对话框中选取 Debug 属性页，在 Executable for debug session 编辑框中输入“c:\matlabr11\bin\matlab.exe”即可，这样就可以对 MEX 文件进行调试了。

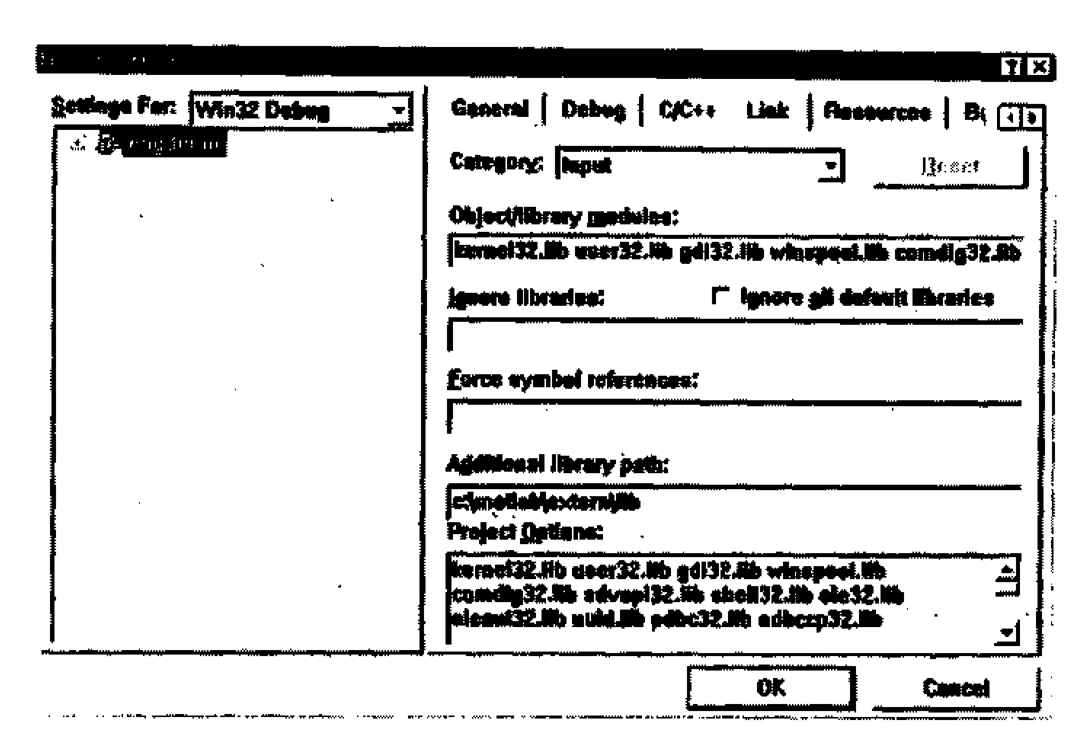


图 3.6　Project Setting Link 对话框

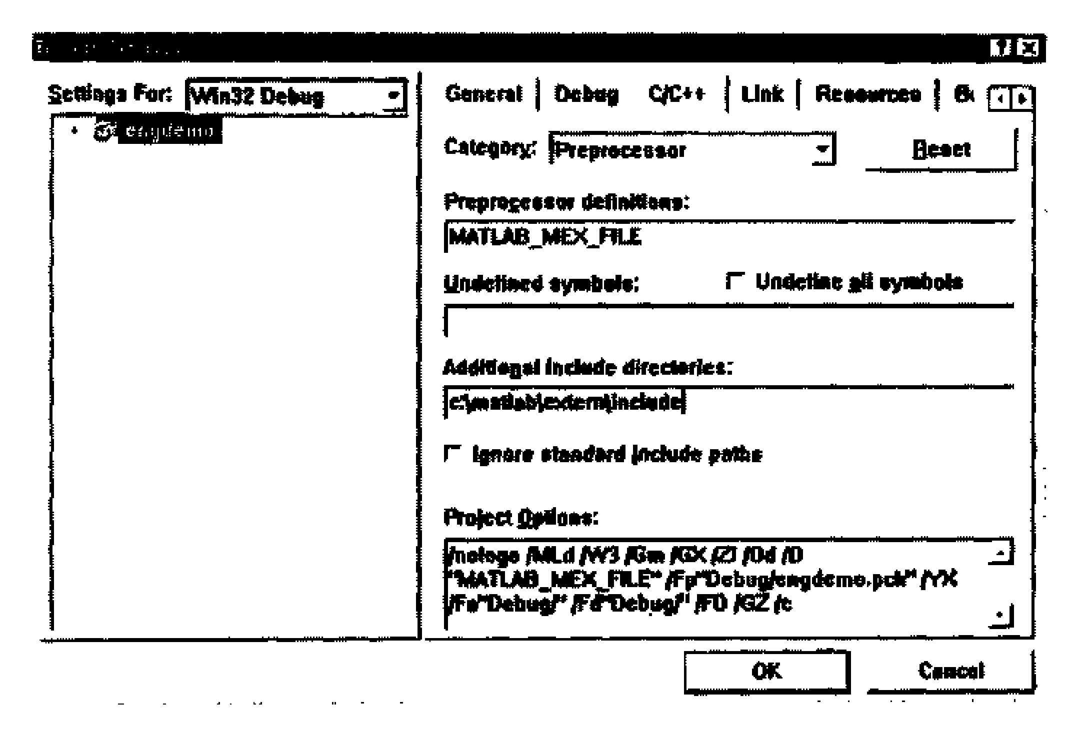


图 3.7　Project Setting C++对话框

3.7　C 语言 mex-函数

C 语言 mex-函数是 MATLAB 应用程序接口函数库提供的一种类型的库函数，它们均以 mex 为前缀，主要功能是与 MATLAB 环境进行交互，例如从 MATLAB 环境中获取必要的阵列数据或者返回一定的信息。这种类型的库函数只能用于 MEX 文件之中，在

本节中，将对这种类型的所有库函数的使用进行详细地说明。

3.7.1　C 语言 mex-函数的声明

在 MATLAB 应用程序接口函数库中，总共提供了 33 个 C 语言的 mex-函数，它们的声明分别如下：

```
void mexAddFlops(int count)
int mexAtExit(void (* ExitFcn)(void))
int mexCallMATLAB(int nlhs, mxArray *plhs[], int nrhs,
                  mxArray *prhs[], const char *command_name)
void mexErrMsgTxt(const char *error_msg)
int mexEvalString(const char *command)
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
const char *mexFunctionName(void)
const mxArray *mexGet(double handle, const char *property)
mxArray *mexGetArray(const char *name, const char *workspace)
const mxArray *mexGetArrayPtr(const char *name, const char *workspace)
mexGetEps (Obsolete)
mexGetFull (Obsolete)
mexGetGlobal (Obsolete)
mexGetInf (Obsolete)
mexGetMatrix (Obsolete)
mexGetMatrixPtr (Obsolete)
mexGetNaN (Obsolete)
mexIsFinite (Obsolete)
bool mexIsGlobal(const mxArray *array_ptr)
mexIsInf (Obsolete)
bool mexIsLocked(void)
mexIsNaN (Obsolete)
void mexLock(void)
void mexMakeArrayPersistent(mxArray *array_ptr)
void mexMakeMemoryPersistent(void *ptr)
int mexPrintf(const char *format, ...)
int mexPutArray(mxArray *array_ptr, const char *workspace)
mexPutFull (Obsolete)
mexPutMatrix (Obsolete)
int mexSet(double handle, const char *property, mxArray *value)
void mexSetTrapFlag(int trap_flag)
void mexUnlock(void)
void mexWarnMsgTxt(const char *warning_msg)
```

所有这些函数均在头文件 mex.h 中予以声明，在使用它们时，必须对该头文件进行包含。

3.7.2　C 语言 mex-函数的使用说明

下面我们将逐一地讲述 C 语言 mex-函数的使用：

1. mexAddFlops

功　能：用于更新 MATLAB 内部的浮点运算计数器。

语　法：#include "mex.h"

void mexAddFlops(int count);

说　明：在 MATLAB 的工作环境中，存在一个浮点运算计数器，用来记录 MATLAB 所进行的浮点运算的次数，但是由于 MEX 文件是用 C 语言编写，所以在 MEX 文件中的所进行的浮点运算的次数并没有被 MATLAB 计数器计数。如果希望 MATLAB 对 MEX 文件中的浮点运算也进行计数，那么必须通过使用函数 mexAddFlops 来手动完成，其输入参数 count 为一个整型数，用于指定 MATLAB 内部浮点运算计数器增加的次数，该函数没有返回值。

举　例：程序 yprime.c 为 MATLAB 提供的一个范例程序，其源代码如下：

```
/* 头文件包含 */
#include <math.h>
#include "mex.h"

/* 输入参数宏定义 */
#define T_IN   prhs[0]
#define Y_IN   prhs[1]

/* 输出参数的宏定义 */
#define YP_OUT plhs[0]

/* 条件编译 */
#if ! defined(MAX)
#define    MAX(A, B)    ((A) > (B) ? (A) : (B))
#endif

#if ! defined(MIN)
#define    MIN(A, B)    ((A) < (B) ? (A) : (B))
#endif

/* 宏定义 */
#define PI 3.14159265
staticdouble    mu = 1/82.45;
staticdouble    mus = 1 - 1/82.45;

/* 计算子例行程序 */
static void yprime(double yp[], double *t, double y[] )
{
    double   r1,r2;

    r1 = sqrt((y[0]+mu)*(y[0]+mu) + y[2]*y[2]);
```

```
    r2 = sqrt((y[0]-mus) * (y[0]-mus) + y[2] * y[2]);

    /* 显示被零除的警告信息 */
    if (r1 == 0.0 || r2 == 0.0 )
    {
        mexWarnMsgTxt("Division by zero! \n");
    }

    yp[0] = y[1];
    yp[1] = 2 * y[3]+y[0]-mus * (y[0]+mu)/(r1 * r1 * r1)-mu
                    * (y[0]-mus)/(r2 * r2 * r2);
    yp[2] = y[3];
    yp[3]=-2 * y[1]+y[2]-mus * y[2]/(r1 * r1 * r1)-mu * y[2]/(r2 * r2 * r2);
    return;
}

/* 入口点函数 */
void mexFunction( int nlhs, mxArray *plhs[],
                          int nrhs, const mxArray *prhs[] )
{
    /* 变量定义 */
    double *yp;
    double *t, *y;
    unsigned int m,n;

    /* 检查输入输出参数个数 */
    if (nrhs != 2)
    {
        mexErrMsgTxt("Two input arguments required.");
    }
    else if (nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 检查变量 Y 的大小,Y 必须为一个包含 4 个元素的行向量或列向量 */
    m = mxGetM(Y_IN);
    n = mxGetN(Y_IN);
    if (! mxIsDouble(Y_IN) || mxIsComplex(Y_IN) ||
            (MAX(m,n) != 4) || (MIN(m,n) != 1))
    {
        mexErrMsgTxt("YPRIME requires that Y be a 4 x 1vector.");
    }

    /* 创建返回矩阵 */
```

```
        YP_OUT = mxCreateDoubleMatrix(m, n, mxREAL);

        /* 获取输出矩阵的实部指针 */
        yp = mxGetPr(YP_OUT);
        t = mxGetPr(T_IN);
        y = mxGetPr(Y_IN);

        /* 调用计算子例行程序 */
        yprime(yp,t,y);

        /* 更新 MATLAB 内部浮点运算计数器 */
        mexAddFlops(38);
        return;
    }
```

2. mexAtExit

功　能：用于登记一个函数，该函数在 MEX 文件被清除或者 MATLAB 终止执行时被调用，用来完成一定的善后工作，如内存释放等。

语　法：#include "mex.h"

int mexAtExit(void (*ExitFcn)(void));

说　明：函数 mexAtExi 的输入参数为一个 void 类型的指针 ExitFcn，用来指向某一个用户函数；其返回值永远为零。使用函数 mexAtExit 允许用户登记注册一个自定义 C 语言函数作为退出函数，该函数在 MEX 文件被清除或者 MATLAB 终止执行时被调用，用来完成一定的善后工作，例如释放持久阵列或者关闭文件等等。对于任意一个 MEX 文件来说，只能登记注册一个处于活动状态的退出函数，当一个函数中使用了多次 mexAtExit 函数时，那么只有其中最后使用的一次为有效。此外当一个 MEX 文件处于锁死状态时，所有的试图清除 MEX 文件的操作都将失败，相应地，当一个用户试图清除一个上锁的 MEX 文件时，MATLAB 将不会自动调用退出函数，即使是在一切操作无误的前提下，有关 MEX 文件的上锁和解锁操作请读者参见函数 mexLock 和函数 mexUnlock。

举　例：MATLAB 中提供了两个范例程序 mexatexit.c 和 mexatexit.cpp，均位于目录

MATLAB 根目录\extern\examples\mex

中，分别对 C 语言和 C++ 语言中函数 mexAtExit 的使用方法进行了讲述，下面我们对它们进行详细地解释说明，以使读者对函数 mexAtExi 的使用有一个全面的认识。程序 mexatexit.c 的功能为打开一个数据文件 matlab.data，并向其中写入字符串，字符串由用户输入，但退出该程序时，调用函数 CloseStream 关闭文件，其源代码如下：

```
/* 程序段(1) */
```

```
#include <stdio.h>
#include "mex.h"

static FILE  *fp=NULL;

/* 程序段(2) */
static void CloseStream(void)
{
   mexPrintf("Closing file matlab.data.\n");
   fclose(fp);
}

/* 程序段(3) */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{

     char *str;

     /* 程序段(4) */
     if (nrhs != 1)
     {
         mexErrMsgTxt("One input argument required.");
     }
     if(nlhs > 1)
     {
         mexErrMsgTxt("Too many output arguments.");
     }
     if (!(mxIsChar(prhs[0])))
     {
         mexErrMsgTxt("Input must be of type string.\n.");
     }

     /* 程序段(5) */
     if (fp==NULL)
     {
         fp = fopen("matlab.data", "w");
         if (fp == NULL)
         {
             mexErrMsgTxt("Could not open file matlab.data.");
         }
         mexPrintf("Opening file matlab.data.\n");

         /* 退出函数注册 */
         mexAtExit(CloseStream);
```

```
        }

        /* 程序段(6) */
        str=mxArrayToString(prhs[0]);

        /* 程序段(7) */
        if (fprintf(fp," %s\n", str) != strlen(str) +1)
        {
            mxFree(str);
            mexErrMsgTxt("Could not write data to file.\n");
        }
        mexPrintf("Writing data to file.\n");
        mxFree(str);
    }
```

其中，程序段(1)完成的功能为对必须的头文件进行包含；程序段(2)为输出函数的定义，其功能为关闭数据文件 matlab.data；程序段(3)为 MEX 文件的入口，此外还定义了一个字符串变量 str；程序段(4)用来对 MEX 文件的输入参数和输出参数的个数及类型进行判断，输入参数的个数必须为 1，类型必须为字符串，输出参数的个数不能大于 1；程序段(5)用来判断数据文件 matlab.data 是否被打开，若没用则以文件属性“w”打开，文件开启成功后，使用函数 mexAtExit 注册函数 CloseStream，用于在 MEX 文件退出时，自动地关闭数据文件；程序段(6)为使用函数 mxArrayToString 从输入参数中获取字符串；程序段(7)将字符串写入数据文件，并且对字符串指针进行内存释放，这是一个必不可少的步骤，因为 MEX 文件的内存自动管理机制并不会释放该段内存，如果不人为地进行释放，将会导致内存的泄漏。

程序 mexatexit.cpp 完成的功能与程序 mexatexit.c 相同，不过使用的方法不同，在程序 mexatexit.cpp 中，使用了类(class)，程序的源代码如下：

```
/* 程序段(1) */
#include <stdio.h>
#include "mex.h"
#include <string.h>

/* 程序段(2) */
class fileresource
{
public:
    fileresource( ) { fp=fopen("matlab.data","w"); }
    ~fileresource( ) { fclose(fp); }
    FILE *fp;
};

static fileresource file;
```

```
/* 程序段(3) */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    char *str;
    /* 程序段(4) */
    if(file.fp == NULL)
    {
        mexErrMsgTxt("Could not open matlab.data\n");
    }

    /* 程序段(5) */
    if (nrhs != 1)
    {
        mexErrMsgTxt("One input argument required.");
    }
    if (nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 程序段(6) */
    if (! (mxIsChar(prhs[0])))
    {
        mexErrMsgTxt("Input must be of type string.\n.");
    }

    /* 程序段(7) */
    str=mxArrayToString(prhs[0]);
    if (fprintf(file.fp,"%s\n", str) != strlen(str) +1)
    {
        mxFree(str);
        mexErrMsgTxt("Could not write data to file.\n");
    }
    mexPrintf("Writing data to file.\n");
    mxFree(str);
    return;
}
```

其中程序段(1)为头文件包含;程序段(2)为类 fileresource 的定义,该类拥有两个成员函数,即构造函数和析构函数,分别用来打开数据文件和关闭数据文件,此外该类还定义了一个文件类型的指针成员变量 fp,用于存放文件句柄,在该程序段的最后,还定义了一个 fileresource 类的类对象 file;程序段(3)为 MEX 文件的入口;程序段(4)为判断数据文件是否打开;程序段(5)用于判断 MEX 文件的输入参数和输出参数的个数是否正确;程序段(6)用于

判断输入参数的类型是否为字符串类型；程序段(7)用于向数据文件中写入用户输入的字符串，并且释放字符串变量 str 所占用的内存。

读者可能会发现这个程序中，并没有包含对 mex-函数 mexAtExit 的调用的语句和对数据文件开启的文件，其实这些工作已经通过C++的类的机制隐式地得到完成。当声明一个类对象时，自动地调用类的构造函数，完成了对数据文件的开启工作，当程序退出时，通过类对象自动地调用析构函数完成了对文件的关闭任务，其功能同函数 mexAtExit 相同。

3. mexCallMATLAB

功　能：用于调用MATLAB 内建函数或用户自定义的MATLAB M 文件以及MEX 文件。

语　法：

```
#include "mex.h"
int mexCallMATLAB(int nlhs, mxArray *plhs[], int nrhs,
                  mxArray *prhs[], const char *command_name);
```

说　明：该函数用于调用一个内建的 MATLAB 数学函数、运算符、M 函数以及其他一些 MEX 文件，调用的函数名、运算符名、M 函数名以及 MEX 文件名通过字符串类型的输入变量 command_name 传递给 MATLAB，其余四个输入参数的含义分别如下：

- nlhs 为所调用的 MATLAB 命令的输出参数的个数；
- plhs 为指向所调用的 MATLAB 命令的输出参数的 mxArray 结构体类型的指针数组；
- nrhs 为所调用的 MATLAB 命令的输入参数的个数；
- prhs 为指向所调用的 MATLAB 命令的输入参数的 mxArray 结构体类型的指针数组；

在默认情况下，调用函数 mexCallMATLAB 执行命令 command_name 发生错误时，MATLAB 将终止程序的运行，并且退回到 MATLAB 命令提示符下；如果用户希望发生错误时，进行另外的处理，可以使用函数 mxSet-TrapFlag 打开陷阱标志，进行异常处理。此外在调用函数 mexCallMATLAB 有可能会产生一个类型为 mxUNKNOWN_CLASS 的数据对象。例如用户定义了一个 M 文件，如下：

```
function [a,b] = double_c(c)
a = 2 * c;
```

该 M 文件拥有两个输出参数，但只有一个在计算过程中予以了赋值，在执行该函数时，MATLAB 将会显示警告信息：

```
Warning: One or more output arguments not assigned during
call to ' double_c '.
```

但是并不报错，而是将未赋值的对象 b 描述为一个空矩阵，如果用户使用函数 mexCallMATLAB 对该函数调用时，这个未赋值的对象将被描述为 mx-

UNKNOWN _CLASS 类型。

举 例：程序 mexcallmatlab.c 为 MATLAB 提供的一个范例程序，它没有任何输入参数，当在 MATLAB 中调用该程序时，它将首先构造并显示下面的矩阵

hankel(1:4,4:-1:1) + sqrt(-1) * toeplitz(1:4,1:4)

然后通过函数 mexCallMATLAB 调用 MATLAB 函数计算矩阵的特征值和特征向量，并对特征值矩阵求逆，最后显示特征向量矩阵。程序的源代码如下：

```
/* 头文件包含 */
#include <math.h>
#include "mex.h"

/* 宏定义 */
#define XR(i,j) xr[i+4*j]
#define XI(i,j) xi[i+4*j]

/* 计算子程序，用于构造矩阵：
   hankel(1:4,4:-1:1) + sqrt(-1) * toeplitz(1:4,1:4)  */

static void fill_array( double *xr, double *xi)
{
    double tmp;
    int i,j,jj;

    /* 填充矩阵的实部和虚部 */
    for (j = 0; j < 4; j++) {
        for (i = 0; i <= j; i++)
        {
            XR(i,j) = 4 + i - j;
            XR(j,i) = XR(i,j);
            XI(i,j) = j - i + 1;
            XI(j,i) = XI(i,j);
        }
    }

    for (j = 0; j < 2; j++)
    {
        for (i = 0; i < 4; i++)
        {
            tmp = XR(i,j);
            jj = 3 - j;
            XR(i,j) = XR(i,jj);
            XR(i,jj) = tmp;
        }
```

```
        }
}

/* 矩阵求逆 */
static void invertd( double *xr, double *xi )
{
    double tmp;
    double *rx, *ix;
    int i;

    rx = xr;
    ix = xi;

    for (i = 0; i < 16; i += 5, rx += 5, ix += 5)
    {
        tmp = *rx * *rx + *ix * *ix;
        *rx = *rx / tmp;
        *ix = - *ix / tmp;
    }
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
    int m, n;
    mxArray *lhs[2], *x;
    m = n = 4;

    /* 检查输入和输出参数的个数 */
    if (nrhs != 0)
    {
        mexErrMsgTxt("No input arguments required.");
    }
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 构造空矩阵 x */
    x = mxCreateDoubleMatrix(m, n, mxCOMPLEX);

    /* 填充矩阵 x */
    fill_array(mxGetPr(x), mxGetPi(x));

    /* 调用函数 mexCallMATLAB 执行 MATLAB 命令 disp,用于显示矩阵 x */
```

```
    mexCallMATLAB(0,NULL,1, &x, "disp");

    /* 调用 MATLAB 函数 eig 执行特征值和特征向量计算任务 */
    mexCallMATLAB(2, lhs, 1, &x, "eig");

    /* 显示特征值矩阵 */
    mexCallMATLAB(0,NULL,1, &lhs[1], "disp");

    /* 特征值矩阵求逆 */
    invertd(mxGetPr(lhs[1]), mxGetPi(lhs[1]));

    /* 显示结果 */
    mexCallMATLAB(0,NULL,1, &lhs[1], "disp");

    /* 释放临时矩阵对象 */
    mxDestroyArray(x);
    mxDestroyArray(lhs[1]);
    plhs[0] = lhs[0];
}
```

该程序执行后,MATLAB 将显示如下计算结果:

```
1.0000 + 1.0000i  2.0000 + 2.0000i  3.0000 + 3.0000i  4.0000 + 4.0000i
2.0000 + 2.0000i  3.0000 + 1.0000i  4.0000 + 2.0000i  3.0000 + 3.0000i
3.0000 + 3.0000i  4.0000 + 2.0000i  3.0000 + 1.0000i  2.0000 + 2.0000i
4.0000 + 4.0000i  3.0000 + 3.0000i  2.0000 + 2.0000i  1.0000 + 1.0000i

10.8990 + 8.8990i          0                  0                  0
      0          -3.4142 - 3.4142i            0                  0
      0                    0          1.1010 - 0.8990i           0
      0                    0                  0         -0.5858 - 0.5858i

  0.0551 - 0.0449i         0                  0                  0
      0          -0.1464 + 0.1464i            0                  0
      0                    0          0.5449 + 0.4449i           0
      0                    0                  0         -0.8536 + 0.8536i

ans =
0.4983+0.0413i  -0.6449+0.1044i   0.4995-0.0227i  -0.2685-0.0332i
0.4965-0.0592i  -0.2671+0.0433i  -0.4939-0.0777i   0.6483+0.0803i
0.4965-0.0592i   0.2671-0.0433i  -0.4939-0.0777i  -0.6483-0.0803i
0.4983+0.0413i   0.6449-0.1044i   0.4995-0.0227i   0.2685+0.0332i
```

4. mexErrMsgTxt

功　能:用于输出错误信息,并返回到 MATLAB 命令提示符下。

语　法:#include "mex.h"

void mexErrMsgTxt(const char *error_msg);

说　明:该函数仅有一个字符串类型的输入参数,没有任何返回值。当程序执行了该函数后,它在 MATLAB 命令窗口中显示一个错误信息 error_msg 后,将终止程序的继续运行,并返回到 MATLAB 命令提示符下。不过这里必须注意一点,虽然对函数 mexErrMsgTxt 的调用会终止当前 MEX 文件的执行,但是并不会将 MEX 文件清除,因此使用函数 mexAtExit 注册的退出函数并不会被调用。不过对于程序中使用 mxCalloc 等函数分配的临时使用的内存,将会被 MEX 文件的内存自动管理机制清除,用户不用担心内存泄漏的问题。

举　例:对函数 mexErrMsgTxt 的使用非常简单,例如

```
mexErrMsgTxt("Too many output arguments.");
```

在前面的例子中该函数得到广泛的使用,往往用于程序发生执行异常时,报告错误并退出程序的运行,读者可以参见前面的各程序。

5. mexEvalString

功　能:用于输入一个表达式命令到 MATLAB 工作环境中执行。

语　法:#include "mex.h"

int mexEvalString(const char *command);

说　明:该函数的输入参数为一个字符串,该字符串代表了一个可以在 MATLAB 工作环境中执行的 MATLAB 命令,如果该命令执行成功,其返回值为 0,若不成功,则返回一个非零的整数值。

该函数与前面讲述的函数 mexCallMATLAB 的功能大致相同,惟一不同的是函数 mexCallMATLAB 不但可以用来执行 MATLAB 命令,而且可以通过参数 nlhs 和参数 plhs 从 MATLAB 环境中得到计算的结果,用于 MEX 文件中的后续计算;而函数 mexEvalString 则没有此项功能,只能用于向 MATLAB 发送计算指令,无法取得计算结果,并且出现于命令右侧的参数必须在当前的 MEX 文件的工作空间中已经存在。

举　例:程序 mexevalstring.c 为 MATLAB 提供的一个使用函数 mexEvalString 的范例程序,该程序没有任何输入参数,其功能为使用函数 mexEvalString 关闭 MATLAB 的报警功能,然后对表达式 0/0 进行计算,在没有关闭 MATLAB 报警功能的情况下,MATLAB 将显示一个被零除的警告,而关闭后,MATLAB 将不显示该警告。程序的源代码如下(程序较为简单,故不作详细解释):

```
/* 头文件包含 */
#include "mex.h"

/* 入口函数 */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
```

```
    /* 检查输入和输出参数的个数 */
    if (nrhs != 0)
    {
        mexErrMsgTxt("No input arguments required.");
    }
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 使用函数 mexEvalString 传送命令 a=0/0,此时 MATLAB 执行该命令将
       报警 */
    mexEvalString("a=0/0");

    /* 关闭报警功能 */
    mexEvalString("warning off");

    /* 在此执行命令 a=0/0,此时 MATLAB 将不报警 */
    mexEvalString("b=0/0");

    /* 开启报警功能 */
    mexEvalString("warning on");
}
```

执行此程序,MATLAB 将显示如下内容:

```
Warning: Divide by zero.
a =
     NaN
b =
     NaN
```

其中 NaN 的含义为 not-a-number。

6. mexFunction

功　能:C 语言 MEX 文件的入口点函数。

语　法:

```
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
                 const mxArray *prhs[]);
```

说　明:该函数拥有四个输入参数,分别为 nlhs、plhs、nrhs 和 prhs,其中 nlhs 为 MEX 文件的输出参数的个数,类型为整数;plhs 为指向输出参数的指针的数组,类型为 mxArray 结构体指针;nrhs 为 MEX 文件的输入参数,类型为整数;prhs 为指向输入参数的指针数组,类型为 mxArray 结构体指针。

函数 mexFunction 是 C 语言 MEX 文件的入口点函数,是任何 C 语言 MEX 文件的必不可少的组成部分,它并非由用户来调用,而是由 MATLAB 系统来使用。当在 MATALB 环境中执行一个 MEX 文件时,MATLAB 首先将

寻找并载入同名的MEX 文件，然后在该文件中寻找名为 mexFunction 的符号名，如果找到，那么 MATLAB 将以该符号的地址作为调用 MEX 文件的入口地址，并且将函数 mexFunction 的各参数自动赋值；如果在整个文件中没有找到名为 mexFunction 的符号，那么 MATLAB 将否认这是一个 MEX 文件并给出相应的错误信息。

举　例： 函数 mexFunction 是 C 语言 MEX 文件的编程基础，深入理解该函数对以后 MEX 文件的编程极为重要。程序 mexfunction.c 是 MATLAB 提供的一个范例程序，它完成的功能相当简单，仅仅是读取输入参数的内容，并原封不动地赋值给输出参数，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"

/* 入口点函数 */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    int i;

    /* 确定输入参数个数，并输出各参数的类型 */
    mexPrintf("\nThere are %d right-hand-side argument(s).", nrhs);
    for (i=0; i<nrhs; i++)
    {
        mexPrintf("\n\tInput Arg %i is of type:\t%s ",i,
                  mxGetClassName(prhs[i]));
    }

    /* 检查输出参数个数，必须小于输入参数个数 */
    mexPrintf("\n\nThere are %d left-hand-side argument(s).\n", nlhs);
    if (nlhs > nrhs)
      mexErrMsgTxt("Cannot specify more outputs than inputs.\n");

    /* 将输入参数赋值给输出参数 */
    for (i=0; i<nlhs; i++)
    {
        plhs[i]=mxCreateDoubleMatrix(1,1,mxREAL);
        *mxGetPr(plhs[i])=mxGetNumberOfElements(prhs[i]);
    }
}
```

7. mexFunctionName

功　能： 用于获得当前执行 MEX 文件的文件名。

语　法： #include "mex.h"

```
const char *mexFunctionName( );
```

说　明：该函数没有任何输入参数，返回值为一个字符串指针，在 MEX 文件中调用该函数时，它将返回当前执行 MEX 文件的文件名，例如在 MATLAB 命令提示符下键入 MEX 文件名 mexname，在该 MEX 文件中存在语句

```
char name[50];
name = mexFunctionName( );
```

这时变量 name 中的内容即为 mexname。

举　例：详细例子参见函数 mexGetArray 的举例。

8. mexGet

功　能：用于获得某一指定图形句柄的属性。

语　法：#include "mex.h"

const mxArray *mexGet(double handle, const char *property);

说　明：输入参数 double handle 为一个指定的图形句柄，参数 const char *property 为图形句柄的属性值。当成功调用该函数时，它返回指定图形句柄的指定属性的属性值；若调用不成功，则返回一个空指针。这里必须注意一点，函数的返回值声明为 const，即常量类型，意味着在对 MEX 文件的操作中，这个 mxArray 结构体类型的指针变量为只读，最好不要对该值进行改动，否则可能产生意想不到的效果。

举　例：程序 mexget.c 是 MATLAB 提供的一个范例程序，它演示了在 MEX 文件中使用函数 mexGet 和函数 mexSet 的方法，其中函数 mexSet 的功能将在后面讲述。该程序的功能为获取一个图形句柄的颜色属性，修改后进行重新设置，其源代码如下：

```
/* 头文件包含 */
include "mex.h"

/* 宏定义 */
#define RED     0
#define GREEN   1
#define BLUE    2

/* MEX 文件入口点函数 */
void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
        const mxArray *prhs[])
{
    /* 变量定义 */
    double          handle;
    const mxArray *color_array_ptr;
    mxArray         *value;
    double          *color;

    /* 程序假定只能拥有一个输入参数，并且该参数的类型必须为 double 类型
```

```
    */
    if(nrhs != 1 || !mxIsDouble(prhs[0]))
    {
        mexErrMsgTxt("Must be called with a valid handle");
    }

    /* 检查输出参数的个数 */
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 确定输入参数是否为一个数量 */
    if (mxGetN(prhs[0]) != 1 || mxGetM(prhs[0]) !=1)
    {
        mexErrMsgTxt("Input must be a scalar handle value.\n");
    }

    /* 获取句柄 */
    handle = mxGetScalar(prhs[0]);

    /* 获取句柄的颜色属性 */
    color_array_ptr = mexGet(handle, "Color");

    /* 判断获取句柄的颜色属性是否成功 */
    if (color_array_ptr == NULL)
        mexErrMsgTxt("Could not get this handle property");

    /* 创建一个颜色属性的拷贝，用于后续的操作，这主要是因为函数 mexGet 的
       返回值为一个常量类型，一般应避免对该返回值的操作 */
    value = mxDuplicateArray(color_array_ptr);

    /* 获取颜色属性的值 */
    color = mxGetPr(value);

    /* 改变颜色值 */
    color[RED] = (1 + color[RED]) /2;
    color[GREEN] = color[GREEN]/2;
    color[BLUE] = color[BLUE]/2;
    /* 重新设置颜色属性 */
    if(mexSet(handle, "Color", value))
        mexErrMsgTxt("Could not set a new 'Color' property.");

    return;
}
```

如果读者对 MATLAB 图形句柄系统不太熟悉，请参见相关的书籍。

9. mexGetArray

功　能：复制另外一个工作空间中的某个变量。

语　法：#include "mex.h"

mxArray *mexGetArray(const char *name, const char *workspace);

说　明：函数输入参数 const char *name 为用户希望从另一个工作空间复制到当前 MEX 文件的工作空间中来的变量的名字，参数 const char *workspace 则指明了变量的来源，它有三种取值，如下：

"base"　：即指明在当前的 MATLAB 工作空间中搜索变量 name；

"caller"　：即指明在调用者的工作空间中搜索变量 name，一般情况下，调用者可以为 MATLAB M 函数、另外的 MEX 文件以及 MATLAB；

"global"　：即指明在所有工作空间中的全局变量中搜索变量 name，如果存在同名变量，但没有标记为全局变量，函数同样会返回 NULL。

如果该函数执行成功，将构造并返回一个 mxArray 结构体类型的指针变量，该指针指向变量 name 的一个拷贝，这时，在我们的 MEX 文件中就可以对变量 name 的数据进行访问和操作了。如果不成功，则返回一个空指针，通常情况下，函数执行失败的原因为当前的工作空间中，并不存在名为 name 的变量。

举　例：程序 mexgetarray.c 为 MATLAB 提供的一个范例程序，它演示了如何在 MEX 文件中使用函数 mexGetArray、函数 mexPutArray 和函数 mexFunctionName，函数 mexPutArray 的功能将在后面进行介绍。该程序的功能非常简单，没有任何输入参数，仅仅用来对当前 MATLAB 环境中 MEX 文件 mexgetarray 被调用的次数进行计数。程序在 MEX 文件的内部和 MATLAB 的内部各设置了一个计数器用来对调用次数进行计数，即使在 MEX 文件被清除后，计数器也不会被清除，这主要是通过设置变量的存储类型来完成。程序的源代码如下：

```
/* 头文件包含 */
#include <stdio.h>
#include <string.h>
#include "mex.h"

/* 定义 MEX 文件内部调用计数器 mex_count，声明为静态变量，作用域为全局
   */
static int mex_count = 0;

/* MEX 文件入口点函数 */
void mexFunction(int nlhs, mxArray *plhs[],
```

```
                int nrhs, const mxArray *prhs[])
{
    /* 变量声明 */
    char array_name[40];
    mxArray *array_ptr;
    int status;

    /* 检查输入输出参数的个数 */
    if (nrhs != 0)
    {
        mexErrMsgTxt("No input arguments required.");
    }
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 调用函数 mexFunctionName 获取 MEX 文件的名字 */
    strcpy(array_name, mexFunctionName( ));

    /* 字符串连接操作 */
        strcat(array_name,"_called");

        /* 在所有的全局变量中搜索名为 array_name 的变量 */
        array_ptr = mexGetArray(array_name,"global");

        /* 检查 MATLAB 和 MEX 文件的计数器状态,并进行相应处理 */
        if (array_ptr == NULL )
        {
            if( mex_count != 0)
            {
            mex_count = 0;
            mexPrintf("Variable %s\n", array_name);
            mexErrMsgTxt ("Global variable was cleared
                        from the MATLAB \global
                        workspace. \nResetting count. \n");
            }

        /* 构造矩阵 */
        array_ptr=mxCreateDoubleMatrix(1,1,mxREAL);

        /* 将阵列 array_ptr 命名为 array_name */
        mxSetName(array_ptr, array_name);
        }
```

```
    /* MATLAB 和 MEX 文件计数器增 1 操作 */
    mex_count=mxGetPr(array_ptr)[0];
    mexPrintf("%s has been called %i time(s)\n",
    mexFunctionName( ), mex_count);

    /* 将变量array_ptr 以"global"的性质输送到 MATLAB
           的工作空间 */
    status = mexPutArray(array_ptr,"global");

    /* 判断输出变量是否正确 */
   if (status==1)
   {
       mexPrintf("Variable %s\n", array_name);
       mexErrMsgTxt("Could not put variable in global workspace. \n");
    }

    /* 析构阵列 */
    mxDestroyArray(array_ptr);
}
```

在 MATLAB 中首次执行该程序时 MATLAB 将显示

```
mexgetarray has been called 1 time(s)
```

而在第二次执行时，计数值将增 1，显示如下：

```
mexgetarray has been called 2 time(s)
```

10. mexGetArrayPtr

功　能：用于获得一个只读的指向另外一个工作空间中的变量的指针。

语　法：#include "mex.h"

const mxArray * mexGetArrayPtr (const char * name, const char * workspace);

说　明：函数输入参数 const char * name 为用户希望从另一个工作空间复制到当前 MEX 文件的工作空间中来的变量的名字，参数 const char * workspace 则指明了变量的来源，它有三种取值，如下：

"base"　：即指明在当前的 MATLAB 工作空间中搜索变量 name；

"caller"　：即指明在调用者的工作空间中搜索变量 name，一般情况下，调用者可以为 MATLAB M 函数、另外的 MEX 文件以及 MATLAB；

"global"　：即指明在所有工作空间中的全局变量中搜索变量 name，如果存在同名变量，但没有标记为全局变量，函数同样会返回 NULL。

如果该函数执行成功，将构造并返回一个 mxArray 结构体类型的指针变量，

该指针指向变量 name 的一个只读属性的拷贝，这对于在 MEX 文件中检查外部某变量的内容极为有用，但是对于数据的操作就无能为力了，这也就是函数 mexGetArrayPtr 与函数 mexGetArray 之间的最大区别。如果函数调用不成功，则返回一个空指针，通常情况下，函数执行失败的原因为当前的工作空间中，并不存在名为 name 的变量。

举　例：在函数 mexGetArray 的范例程序 mexgetarray.c 中，将如下语句

```
array_ptr = mexGetArray(array_name,"global");
```

改为

```
mxArray *array_readonly;
array_readonly = mexGetArrayPtr(array_name,"global");
array_ptr = mxDuplicateArray(array_readonly);
```

可以实现同样功能。更为详细的例子参见函数 mxIsLogical 的范例程序。

11. mexGetEps(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mxGetEps 来代替函数 mexGetEps，例如在 MEX 文件中可以使用语句

```
eps = mxGetEps( );
```

代替对函数 mexGetEps 的调用语句

```
eps = mexGetEps( );
```

如果需要使用已经存在对函数 mexGetEps 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。详细参见函数 mxGetEps。

12. mexGetFull(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。为了完成同样的功能，在 MATLAB V5.0 以上的版本中，可以使用如下语句

```
mexGetArray(array_ptr, "caller");
name = mxGetName(array_ptr);
m = mxGetM(array_ptr);
n = mxGetN(array_ptr);
pr = mxGetPr(array_ptr);
pi = mxGetPi(array_ptr);
```

代替语句

```
mexGetFull(name, m, n, pr, pi);
```

如果需要使用已经存在对函数 mexGetFull 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。
详细内容可以参阅相关的函数 mexGetArray、mxGetName、mxGetPr 和函数 mxGetPi。

13. mexGetGlobal(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mexGetArrayPtr 来代替函数 mexGetGlobal，例如在 MEX 文件中可以使用语句

```
mexGetArrayPtr(name, "global");
```

代替对函数 mexGetGlobal 的调用语句

```
mexGetGlobal(name);
```

如果需要使用已经存在对函数 mexGetGlobal 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。
详细内容可以参阅相关的函数 mexGetArray，mxGetName，mxGetPr 和函数 mxGetPi。

14. mexGetInf(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mxGetInf 来代替函数 mexGetInf，例如在 MEX 文件中可以使用语句

```
Inf = mxGetInf ( );
```

代替对函数 mexGetInf 的调用语句

```
Inf = mexGetInf ( );
```

如果需要使用已经存在对函数 mexGetInf 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。
详细内容可以参阅相关的函数 mxGetInf。

15. mexGetMatrix(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mexGetArray 来代替函数 mexGetMatrix，例如在 MEX 文件中可以使用语句

```
mexGetArray(name, "caller");
```

代替对函数 mexGetMatrix 的调用语句

```
mexGetMatrix(name);
```

如果需要使用已经存在对函数 mexGetMatrix 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。
详细内容可以参阅相关的函数 mexGetArray。

16. mexGetMatrixPtr(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mexGetArrayPtr 来代替函数 mexGetMatrixPtr，例如在 MEX 文件中可以使用语句

```
mexGetArrayPtr(name, "caller");
```

代替对函数 mexGetMatrixPtr 的调用语句

```
mexGetMatrixPtr(name);
```

如果需要使用已经存在对函数 mexGetMatrixPtr 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。
详细内容可以参阅相关的函数 mexGetArrayPtr。

17. mexGetNaN(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mxGetNaN 来代替函数 mexGetNaN，例如在 MEX 文件中可以使用语句

```
NaN = mxGetNaN();
```

代替对函数 mexGetNaN 的调用语句

```
NaN = mexGetNaN( );
```

如果需要使用已经存在对函数 mexGetNaN 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

详细内容可以参阅相关的函数 mxGetNaN。

18. mexIsFinite(*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mxIsFinite 来代替函数 mexIsFinite，例如在 MEX 文件中可以使用语句

```
answer = mxIsFinite(value);
```

代替对函数 mexIsFinite 的调用语句

```
answer = mexIsFinite(value);
```

如果需要使用已经存在对函数 mexIsFinite 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

详细内容可以参阅相关的函数 mxIsFinite。

19. mexIsGlobal

功　能：判断 mxArray 结构体类型变量是否为全局作用域。

语　法：

```
#include "mex.h"
bool mexIsGlobal(const mxArray *array_ptr);
```

说　明：该函数的输入参数为一个指向 mxArray 结构体变量的指针，当函数判断指针 array_ptr 所指向的 mxArray 结构体变量为全局作用域时，将返回逻辑真，否则返回逻辑假。

使用该函数可以判断一个指定的 mxArray 结构体变量是否拥有全局作用域，在默认情况下，MEX 文件中和独立运行的程序中所有的 mxArray 结构体变量作用域均为局部，这意味着所在文件和程序内部对 mxArray 结构体变量所作的修改均不会对别的工作空间中的同名 mxArray 结构体变量产生影响；而拥有全局作用域的 mxArray 结构体变量进行修改，则完全不同，在一个工作空间中的动作，将对所用工作空间中的同名变量产生影响。在 MATLAB 工作环境中可以使用命令 global 将一个变量赋予全局作用域，例如命令

```
global x;
```

的含义为赋予变量 x 全局作用域。函数 mexIsGlobal 最常见的用途是用来判断一个 MAT 文件中的变量是否为全局变量。

举　例：参见函数 mxislogical. c 的范例程序。

20. mexIsInf(obsolete)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mxIsInf 来代替函数 mexIsInf，例如在 MEX 文件中可以使用语句

```
answer = mxIsInf(value);
```

代替对函数 mexIsInf 的调用语句

```
answer = mexIsInf(value);
```

如果需要使用已经存在对函数 mexIsInf 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

详细内容可以参阅相关的函数 mxIsInf。

21. mexIsLocked

功　能：判断 MEX 文件是否处于锁定状态。

语　法：

```
#include "mex.h"
bool mexIsLocked(void);
```

说　明：当 MEX 文件被锁定时，函数返回逻辑真；否则函数返回逻辑假。当一个 MEX 文件被锁定时，用户不可以将该文件从内存中删除。

举　例：参见函数 mexLock 的范例程序。

22. mexIsNaN(obsolete)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mxIsNaN 来代替函数 mexIsNaN，例如在 MEX 文件中可以使用语句

```
answer = mxIsNaN(value);
```

代替对函数 mexIsNaN 的调用语句

```
answer = mexIsNaN(value);
```

如果需要使用已经存在对函数 mexIsNaN 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参

数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

详细内容可以参阅相关的函数 mxIsNaN。

23. mexLock

功 能：用于锁定一个 MEX 文件。

语 法：#include "mex.h"

void mexLock(void)；

说 明：该函数没有任何输入和输出参数。在默认情况下，所有的 MEX 文件均处于解锁状态，这意味着用户可以在任意的时刻将 MEX 文件从内存中清除，但是当对 MEX 文件使用函数 mexLock 进行加锁后，就可以禁止 MEX 文件被随意清除。同时可以对同一个 MEX 文件多次使用函数 mexLock 进行加锁，这时，函数 mexLock 将锁定次数保存在一个 Lock 计数器中，并且每调用一次函数 mexLock，该计数器增 1。对 MEX 文件的解锁可以使用函数 mexUnlock，当对文件多次加锁时，必须多次使用函数 mexUnlock 进行解锁。函数 mexUnlock 将在后面进行讲述。

举 例：程序 mexlock.c 为 MATLAB 提供的一个范例程序，它显示了如何在 MEX 文件中使用函数 mexLock、函数 mexUnlock 和函数 mexIsLocked。其功能非常简单，即根据用户的输入，对 MEX 文件进行一定的加锁和解锁操作，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"

/* MEX 文件入口点函数 */
void mexFunction(int nlhs,mxArray *plhs[],
                 int nrhs,const mxArray *prhs[])
{
    /* 定义 double 型变量 lock，当 lock 取值为 1 时，表示希望对文件加锁；当 lock
    取值为 0 时，表示对 MEX 文件保持原状态不动；当 lock 取值为 -1 时，表示希
    望对文件进行解锁操作。*/
    double lock;

    /* 对输入变量进行判断，输入参数的个数必须为一个，类型必须为非复数的
    数量 */
    if (nrhs != 1 || ! mxIsDouble(prhs[0]) ||
                    mxGetN(prhs[0]) * mxGetM(prhs[0]) != 1 ||
                    mxIsComplex(prhs[0]))
    {
        mexErrMsgTxt("Input argument must be a real scalar double");
    }

    /* 不允许有输出参数 */
    if(nlhs > 0)
```

```
{
    mexErrMsgTxt("No output arguments expected.");
}

/* 获取输入参数的内容 */
lock = mxGetScalar(prhs[0]);

/* 限定变量 lock 取值范围为 0,1,-1 */
if((lock != 0.0) && lock != 1.0 && lock != -1.0)
{
    mexErrMsgTxt("Input argument must be either 1 to lock or -1 to\
                unlock or 0 for lock status.\n");
}

/* 判断文件的状态,即解锁或加锁,然后根据输入进行相应的操作 */
if(mexIsLocked( ))
{
    if(lock > 0.0)
    {
        mexErrMsgTxt("MEX-file is already locked\n");
    }
    else if(lock < 0.0)
    {
        /* 文件解锁 */
        mexUnlock( );
        mexPrintf("MEX-file is unlocked\n");
    }
    else
    {
        mexPrintf("MEX-file is locked\n");
    }
}
else
{
    if(lock < 0.0)
    {
        mexErrMsgTxt("MEX-file is already unlocked\n");
    }
    else if(lock > 0.0)
    {
        /* 文件加锁 */
        mexLock( );
        mexPrintf("MEX-file is locked\n");
    }
    else
```

```
            {
                mexPrintf("MEX-file is unlocked\n");
            }
        }
        return;
    }
```

24. mexMakeArrayPersistent

功　能：将一个阵列转换为持久阵列对象。

语　法：#include "mex.h"

void mexMakeArrayPersistent(mxArray *array_ptr);

说　明：函数的输入参数 mxArray *array_ptr 为一个指向由函数 mxCreate 构造的 mxArray 结构体类型数据的指针。在一般情况下，由函数 mxCreate 在 MEX 文件中动态创建的 mxArray 结构体类型数据均为临时变量，在 MEX 文件执行完毕退出时，这些变量都将被自动清除(原理请读者参阅 3.3 节：MEX 文件的内存管理)。如果用户希望某个阵列在 MEX 文件执行结束后仍然存在，方法之一就是使用函数 mexMakeArrayPersistent 将该阵列转换为一个持久阵列对象(参见 3.3 节)，这种数据对象在 MEX 文件退出时不会被自动内存管理机制所释放。但是在使用持久阵列对象时必须非常小心，必须人为清除该对象，否则将会导致内存的泄漏，一般的方法是使用函数 mexAtExit 注册一个自定义函数用于清除持久阵列对象。

举　例：使用实例请读者参见 3.3.2 小节中的 MEX 文件。

25. mexMakeMemoryPersistent

功　能：将一个内存段转换为持久阵列对象。

语　法：#include "mex.h"

void mexMakeMemoryPersistent(void *ptr);

说　明：函数输入参数 void *ptr 为一个指向某一内存段的指针，该内存段一般由函数 mxCalloc、函数 mxMalloc 和函数 mxRealloc 进行分配。一般情况下，由这些函数分配的内存将在 MEX 文件退出时，被自动清除(原理请读者参阅 3.3 节：MEX 文件的内存管理)。如果用户希望某一段内存被免于自动清除，方法之一是使用函数 mexMakeMemoryPersistent 将该段内存转换为持久阵列对象(参见 3.3 节)。当用户声明一段内存为持久阵列对象后，必须人为地将之清除，比如使用函数 mexAtExit 注册一个自定义函数用于清除工作。

举　例：下面是一个使用函数 mexMakeMemoryPersistent 的非常简单的 MEX 文件，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"

/* 全局变量定义 */
```

```
static int initialized = 0;
char *buf;

/* 持久阵列对象清除函数 */
void cleanup(void)
{
    mexPrintf("MEX-file is terminating, destroying memory\n");
    mxFree(buf);
}

/* 入口子例行程序 */
void mexFunction(int nlhs,mxArray *plhs[],
                 int nrhs,const mxArray *prhs[])
{

    if (! initialized)
    {
        mexPrintf("MEX-file initializing, allocate memory\n");

        /* 创建持久阵列对象 */
        buf = (char *) mxCalloc(50, sizeof(char));
        mexMakeMemoryPersistent(buf);

        /* 注册 clearup 函数 */
        mexAtExit(cleanup);

        initialized = 1;
    }
    else
    {
        mexPrintf("MEX-file initialized \n");
    }
}
```

它的功能为声明一个内存段为持久阵列对象，并使用函数 mexAtExit 注册了一个函数在 MATLAB 终止执行时清除该持久阵列对象。

26. mexPrintf

功　能：输出。

语　法：#include "mex.h"

int mexPrintf(const char *format, ...);

说　明：该函数的功能及函数参数与 C 语言的输出函数 printf 的功能及函数参数完全相同，读者可以参见 C 语言中对输出格式的说明。在 MATLAB 中，C 语言函数 printf 已经内嵌入 MATLAB 中，当用户调用函数 mexPrintf 时，MATLAB 将会回调 C 语言函数 printf 来完成输出功能。但是在 MEX 文件中，用

户必须使用函数 mexPrintf 来代替 C 语言函数 printf 来完成输出任务。

举　例：mexPrintf("total num of cells = %d\n", total_num_of_cells);

27. mexPutArray

功　能：将当前 MEX 文件工作空间中的某个变量输出到另外一个工作空间中。

语　法：#include "mex.h"

int mexPutArray(mxArray *array_ptr, const char *workspace);

说　明：函数输入参数 mxArray *array_ptr 为用户希望从当前 MEX 文件的工作空间中输出的 mxArray 结构体的指针，参数 const char *workspace 则指明了阵列的去处，即目的工作空间，它有三种取值，如下：

"base"　：即指明将 array_ptr 所指向的 mxArray 结构体变量输出到当前的 MATLAB 工作空间中；

"caller"　：即指明将 array_ptr 所指向的 mxArray 结构体变量输出到调用该 MEX 文件的工作空间中，一般情况下，调用者可以为 MATLAB M 函数、另外的 MEX 文件以及 MATLAB；

"global"　：即指明将 array_ptr 所指向的 mxArray 结构体变量输出到全局变量列表。

如果该函数执行成功，将返回 0，否则返回 1。一般情况下，函数执行失败有两种可能性，第一种为指针 array_ptr 为空(NULL)，第二种为指针 array_ptr 所指向的 mxArray 结构体变量，变量没有名字，这时可以使用函数 mxSetName 对该变量设置名字。

通过使用函数 mexPutArray，可以使当前 MEX 文件工作空间中的 mxArray 对象从其他的工作空间所获得，如 MATLAB、M 文件以及另外的 MEX 文件。这里必须非常注意一点，即指针 array_ptr 和指针 array_ptr 指向的 mxArray 结构体变量的名字的区别，在 MEX 文件中，程序通过指针 array_ptr 对 mxArray 结构体变量进行访问；而当将 mxArray 结构体变量输出到 MATLAB 中后，MATLAB 对变量的访问则是通过结构体变量的名字，这也就是为什么如果不对结构体变量设置名字函数执行将失败的原因。如果在目的工作空间中已经存在了同名的变量，那么函数将覆盖现存的变量。例如用户在 MATLAB 的工作空间中已经定义了变量 peach 为

```
peach = 1 :4;
```

然后以如下语句调用函数 mexPutArray，

```
mxSetName(array_ptr, "peach");
mexPutArray(array_ptr, "base");
```

这时指针 array_ptr 指向的 mxArray 结构体变量将覆盖原变量 peach。

举　例：参见函数 mexGetArray 的范例程序。

28. mexPutFull(obsolete)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保

持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。为了完成同样的功能，在 MATLAB V5.0 以上的版本中，可以使用如下语句

```
array_ptr = mxCreateDoubleMatrix(0, 0, mxREAL/mxCOMPLEX);
mxSetName(array_ptr, name);
mexPutArray(array_ptr, "caller");
```

代替语句

```
mexPutFull(name, m, n, pr, pi);
```

如果需要使用已经存在对函数 mexPutFull 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

29. mexPutMatrix(obsolete)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MEX 文件中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 mexPutArray 来代替函数 mexPutMatrix，例如在 MEX 文件中可以使用语句

```
mexPutArray(array_ptr, "caller");
```

代替对函数 mexPutMatrix 的调用语句

```
mexPutMatrix(matrix_ptr);
```

如果需要使用已经存在对函数 mexPutMatrix 调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

详细内容可以参阅相关的函数 mexPutArray。

30. mexSet

功　能：用于设置某个图形句柄的属性值。

语　法：

```
#include "mex.h"
int mexSet(double handle, const char *property, mxArray *value);
```

说　明：该函数与函数 mexGet 相对应，用于设置某个图形句柄的属性值，输入参数 handle 代表了一个特定的图形句柄，property 代表了句柄的某个属性值，而 value 代表了将要设置的值。若函数执行成功，将返回 0，否则返回 1。导致不成功的原因可能有：一，没有指定的属性值；二，设定的属性质类型错误或越界。总体来说，该函数的功能与 MATLAB 内建函数 set 的功能大致相同。

举　例：参见函数 mexGet 的范例程序。

31. mexSetTrapFlag

功　能：异常处理。

语　法：#include "mex.h"

void mexSetTrapFlag(int trap_flag)；

说　明：使用函数 mexSetTrapFlag 可以控制调用函数 mexCallMATLAB 失败时 MATLAB 的相应动作。如果用户程序中没有使用函数 mexSetTrapFlag，MATLAB 探测到在执行函数 mexCallMATLAB 时发生错误，则立即终止 MEX 文件的执行并返回到 MATLAB 命令提示符下；如果使用了函数 mexSetTrapFlag，且将输入参数 trap_flag 设置为 0，则效果相同；如果使用了函数 mexSetTrapFlag 并且将参数 trap_flag 设置为 1，那么在探测到发生错误时，MATLAB 将不会终止 MEX 文件的执行，而是返回到调用函数 mexCallMATLAB 的语句的下一行语句，这样用户就可以在 MEX 文件中进行相应的处理了。到目前为止，函数 mexSetTrapFlag 的输入参数 trap_flag 的取值只能为 0 和 1。

举　例：程序 mexsettrapflag.c 是 MATLAB 提供的一个演示函数 mexSetTrapFlag 使用的范例程序，

```
/* 头文件包含 */
#include "mex.h"

/* MEX 文件入口点函数 */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    int status;

    /* 检查输入参数的个数及类型 */
    if (nrhs != 1 || !mxIsDouble(prhs[0]) ||
            mxGetN(prhs[0]) * mxGetM(prhs[0]) != 1 ||
            mxIsComplex(prhs[0]))
    {
        mexErrMsgTxt("Input argument must be a real scalar double.");
    }

    /* 检查输出参数个数 */
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 从输入参数获取函数 mexTrapFlag 的输入参数，进行设置 */
    mexSetTrapFlag((int)mxGetScalar(prhs[0]));
```

```
        /* 激活一个 MATLAB 命令 */
        status = mexCallMATLAB(0, (mxArray **)NULL,
                0, (mxArray **)NULL, "nofcn");

        /* 如果用户输入得到该程序的参数为 1,则当函数 mexCallMATLAB 调用发
           生错误时,控制仍然存在于 MEX 文件中 */
        mexPrintf("mexCallMATLAB status = %d\n", status);
        mexErrMsgTxt("mexCallMATLAB failed. \n");
    }
```

32. mexUnlock

功　能:对 MEX 文件进行解锁。

语　法:#include "mex.h"

void mexUnlock(void);

说　明:在默认情况下,所有的 MEX 文件均不加锁,用户可以在任意情况下将 MEX 删除,如果对文件使用函数 mexLock 进行加锁后,则必须使用函数 mexUnlock 进行解锁后才能清除。如果对一个文件进行多次加锁,则必须相应地多次解锁。

举　例:参见函数 mexLock 的范例程序。

33. mexWarnMsgTxt

功　能:发出警告信息。

语　法:#include "mex.h"

void mexWarnMsgTxt(const char *warning_msg);

说　明:该函数的使用将导致 MATLAB 显示一条警告信息,信息内容由函数输入参数 const char *warning_msg 决定,不同于函数 mexErrMsgTxt,在该函数执行完毕后,并不终止当前 MEX 文件的运行。

举　例:mexWarnMsgTxt("data type is different. ")

3.8　C 语言 mx-函数

3.8.1　C 语言 mx-函数的声明

在 MATLAB 应用程序接口函数库中,总共提供了 93 个 C 语言的 mx-函数,它们的声明分别如下:

```
char *mxArrayToString(const mxArray *array_ptr)
void mxAssert(int expr, char *error_message)
void mxAssertS(int expr, char *error_message)
int mxCalcSingleSubscript(const mxArray *array_ptr, int nsubs, int *subs)
```

```
void *mxCalloc(size_t n, size_t size)
void mxClearLogical(mxArray *array_ptr)
typedef enum mxComplexity(mxREAL=0, mxCOMPLEX)
mxArray *mxCreateCellArray(int ndim, const int *dims)
mxArray *mxCreateCellMatrix(int m, int n)
mxArray *mxCreateCharArray(int ndim, const int *dims)
mxArray *mxCreateCharMatrixFromStrings(int m, const char **str)
mxArray *mxCreateDoubleMatrix(int m, int n, mxComplexity ComplexFlag)
mxCreateFull (Obsolete)
mxArray *mxCreateNumericArray(int ndim, const int *dims,
                              mxClassID class, mxComplexity ComplexFlag)
mxArray *mxCreateSparse(int m, int n, int nzmax, mxComplexity ComplexFlag)
mxArray *mxCreateString(const char *str)
mxArray *mxCreateStructArray(int ndim, const int *dims,
                             int nfields,const char **field_names)
mxArray *mxCreateStructMatrix(int m, int n, int nfields, const char **field_names)
void mxDestroyArray(mxArray *array_ptr)
mxArray *mxDuplicateArray(const mxArray *in)
mxArray *mxFree(void *ptr)
mxFreeMatrix (Obsolete)
mxArray *mxGetCell(const mxArray *array_ptr, int index)
mxClassID mxGetClassID(const mxArray *array_ptr)
const char *mxGetClassName(const mxArray *array_ptr)
void *mxGetData(const mxArray *array_ptr)
const int *mxGetDimensions(const mxArray *array_ptr)
int mxGetElementSize(const mxArray *array_ptr)
double mxGetEps(void)
mxArray *mxGetField(const mxArray *array_ptr, int index, const char *field_name)
mxArray *mxGetFieldByNumber(const mxArray *array_ptr, int index, int field_number)
const char *mxGetFieldNameByNumber(const mxArray *array_ptr, int field_number)
int mxGetFieldNumber(const mxArray *array_ptr, const char *field_name)
void *mxGetImagData(const mxArray *array_ptr)
double mxGetInf(void)
int *mxGetIr(const mxArray *array_ptr)
int *mxGetJc(const mxArray *array_ptr)
int mxGetM(const mxArray *array_ptr)
int mxGetN(const mxArray *array_ptr)
const char *mxGetName(const mxArray *array_ptr)
double mxGetNaN(void)
int mxGetNumberOfDimensions(const mxArray *array_ptr)
int mxGetNumberOfElements(const mxArray *array_ptr)
int mxGetNumberOfFields(const mxArray *array_ptr)
int mxGetNzmax(const mxArray *array_ptr)
double *mxGetPi(const mxArray *array_ptr)
double *mxGetPr(const mxArray *array_ptr)
```

```
double mxGetScalar(const mxArray *array_ptr)
int mxGetString(const mxArray *array_ptr, char *buf, int buflen)
bool mxIsCell(const mxArray *array_ptr)
bool mxIsChar(const mxArray *array_ptr)
bool mxIsClass(const mxArray *array_ptr, const char *name)
bool mxIsComplex(const mxArray *array_ptr)
bool mxIsDouble(const mxArray *array_ptr)
bool mxIsEmpty(const mxArray *array_ptr)
bool mxIsFinite(double value)
bool mxIsFromGlobalWS(const mxArray *array_ptr)
mxIsFull (Obsolete)
bool mxIsInf(double value)
bool mxIsInt8(const mxArray *array_ptr)
bool mxIsInt16(const mxArray *array_ptr)
bool mxIsInt32(const mxArray *array_ptr)
bool mxIsLogical(const mxArray *array_ptr)
bool mxIsNaN(double value)
bool mxIsNumeric(const mxArray *array_ptr)
bool mxIsSingle(const mxArray *array_ptr)
bool mxIsSparse(const mxArray *array_ptr)
mxIsString (Obsolete)
bool mxIsStruct(const mxArray *array_ptr)
bool mxIsUint8(const mxArray *array_ptr)
bool mxIsUint16(const mxArray *array_ptr)
bool mxIsUint32(const mxArray *array_ptr)
void *mxMalloc(size_t n)
void *mxRealloc(void *ptr, size_t size)
void mxSetAllocFcns(calloc_proc callocfcn, free_proc freefcn,
                    realloc_proc reallocfcn, malloc_proc mallocfcn)
void mxSetCell(mxArray *array_ptr, int index, mxArray *value)
int mxSetClassName(mxArray *array_ptr, const char *classname)
void mxSetData(mxArray *array_ptr, void *data_ptr)
int mxSetDimensions(mxArray *array_ptr, const int *dims, int ndims)
void mxSetField(mxArray *array_ptr, int index, const char *field_name, mxArray *value)
void mxSetFieldByNumber(mxArray *array_ptr, int index,
                        int field_number, mxArray *value)
void mxSetImagData(mxArray *array_ptr, void *pi)
void mxSetIr(mxArray *array_ptr, int *ir)
void mxSetJc(mxArray *array_ptr, int *jc)
void mxSetLogical(mxArray *array_ptr)
void mxSetM(mxArray *array_ptr, int m)
void mxSetN(mxArray *array_ptr, int n)
void mxSetName(mxArray *array_ptr, const char *name)
void mxSetNzmax(mxArray *array_ptr, int nzmax)
void mxSetPi(mxArray *array_ptr, double *pi)
```

```
void mxSetPr(mxArray *array_ptr, double *pr)
```

所有这些函数均在头文件 matrix.h 中予以声明，在使用它们时，必须对该头文件进行包含。

3.8.2　C 语言 mx-函数的使用说明

1. mxArrayToString

功　能：将 mxArray 结构体转化为字符串。

语　法：#include "matrix.h"

char *mxArrayToString(const mxArray *array_ptr);

说　明：通过函数 mxArrayToString，用户可以将一个字符串类型的 mxArray 结构体对象中的字符读取出来，并转存为一个 C 语言风格的字符串，该字符串以空字符结尾。如果函数执行成功，函数将返回指向字符串起始位置的字符指针，字符串所占用的内存区域通过使用函数 mxCalloc 来动态分配。在程序结束前，必须通过使用函数 mxFree 来释放，否则将导致内存泄漏。

举　例：参见函数 mexAtExit 的范例程序 mexatexit.c。

2. mxAssert

功　能：出于调试的目的检验某个表达式的值，在出错时，输出错误信息。

语　法：#include "matrix.h"

void mxAssert(int expr, char *error_message);

说　明：该函数的功能非常类似于标准 C 语言的 assert( )宏，通过该函数，用户可以对某个表达式的值进行判断，如果满足要求，则程序继续执行，否则程序终止运行，并且输出指定的信息。使用该函数，可以替代大量的条件判断语句，并且这些语句仅在调试时有效，用户可以在不修改源代码的前提下，通过设置编译参数，在发布的应用程序中，去除这些语句的作用。在程序中大量使用该函数，是一种良好的编程习惯，通过该函数，用户可以使自己减少代码的误用，并且可以检查出逻辑错误的传播。函数的输入参数的含义如下：

- expr 为一个表达式，当该表达式取值为真时，程序继续执行，否则程序终止；
- error_message 为一个错误信息，在表达式 expr 为假时，输出到屏幕上，以提示用户错误的原因。

举　例：下面是一个简单的样例程序，源代码如下：

```
/* 头文件包含 */
#include <stdio.h>
#include "matrix.h"
#include <string.h>

/* 子函数的原型声明 */
```

```
void analyze _string( char *string );

/* 主函数 */
void main( void )
{
    /* 变量定义 */
    char test1[] = "abc", *test2 = NULL, test3[] = "";

    printf ( "Analyzing string '%s'\n", test1 );
    analyze _string( test1 );
    printf ( "Analyzing string '%s'\n", test2 );
    analyze _string( test2 );
    printf ( "Analyzing string '%s'\n", test3 );
    analyze _string( test3 );
}

/* 测试字符串是否为 NULL、空或者长度小于 2 */
/* empty, or longer than 0 characters */
void analyze _string( char * string )
{
    /* 测试字符串是否为 NULL */
    mxAssert( string ! = NULL );

    /* 测试字符串是否为空 */
    mxAssert( *string ! = '\0' );

    /* 测试字符串长度是否小于 2 */
    mxAssert( strlen( string ) > 2 );
}
```

3. mxAssertS

功　能：出于调试的目的检验某个表达式的值，但是在出错时，不输出错误信息。

语　法：#include "matrix.h"

void mxAssertS(int expr, char *error _message);

说　明：函数 mxAssertS 的功能大致与函数 mxAssert 的功能相同，惟一不同是函数 mxAssertS 在表达式 expr 为错时，仅仅是终止程序的运行，而不输出错误信息。

举　例：下面是一个简单的样例程序，源代码如下：

```
/* 头文件包含 */
#include <stdio.h>
#include "matrix.h"
#include <string.h>
```

```
/* 子函数的原型声明 */
void analyze_string( char *string );

/* 主函数 */
void main( void )
{
    /* 变量定义 */
    char test1[] = "abc", *test2 = NULL, test3[] = "";
    printf ( "Analyzing string '%s'\n", test1 );
    analyze_string( test1 );
    printf ( "Analyzing string '%s'\n", test2 );
    analyze_string( test2 );
    printf ( "Analyzing string '%s'\n", test3 );
    analyze_string( test3 );
}

/* 测试字符串是否为NULL、空或者长度小于 2 */
/* empty, or longer than 0 characters */
void analyze_string( char * string )
{
    /* 测试字符串是否为NULL */
    mxAssertS( string != NULL );

    /* 测试字符串是否为空 */
    mxAssertS( *string != '\0' );
    /* 测试字符串长度是否小于 2 */
    mxAssertS( strlen( string ) > 2 );
}
```

4. mxCalcSingleSubscript

功　能:获取指定元素的索引值。

语　法:#include <matrix.h>

int mxCalcSingleSubscript(const mxArray *array_ptr, int nsubs, int *subs);

说　明:通过函数 mxCalcSingleSubscript,用户可以获得 mxArray 结构体中指定元素的索引值,也就是从 mxArray 结构体的起始元素到指定元素间的偏移量,例如 mxArray 结构体的第一个元素的索引值为 0,而最后一个元素的索引值为N-1,其中 N 为mxArray 结构体元素的总的个数。这里必须注意的一点是,在 mxArray 结构体的内部,所有的数据元素均以列优先的原则存放为一个一维的数组,而无论原来 mxArray 结构体对象的维数为多少,这与 MATLAB 中阵列的数据存储方法是一致的。如果函数执行成功,将返回 mxArray 结构体中指定元素的索引值,此外函数的三个输入参数的含义分别如下:

- array_ptr 为一个指向 mxArray 结构体对象的指针;

- nsubs 为一个整型变量，用于指定数组 subs 中元素的个数，一般将该值设定为 mxArray 结构体对象的维数；
- subs 为一个具有 nsubs 个元素的数组，其中 subs[0]代表了 mxArray 结构体对象中指定元素的行数，subs[1]代表了 mxArray 结构体对象中指定元素的列数，subs[2]代表了 mxArray 结构体对象中指定元素的页面数。这里必须注意的一点是，在 MATLAB 中，行和列的起始数目为 1，而在 C 语言中，数组行列的起始均为 0，所以如果要表示 MATLAB 阵列中的第一个元素(1,1)，应将 subs[0] 和 subs[1]均设置为 0。

举　例：程序 mxcalcsinglesubscript.c 是 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中。其功能相当简单，当用户输入一个双精度类型的阵列并指定阵列中的某个元素，通过该程序就可以获得该元素的索引值。函数的源代码如下，我们将在程序中对各语句的功能进行注释：

```
/* mex.h 头文件包含，对于 MEX 文件，这是必不可少的 */
#include "mex.h"

/* MEX 文件的入口点函数 */
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{

    /* 变量说明 */
    int       nsubs, index, x;
    double    *temp;
    int       *subs;

    /* 检查输入变量的个数，要求必须为 2 */
    if (nrhs != 2)
    {
        mexErrMsgTxt("Two input arguments required.");
    }

    /* 检查输出变量的个数，要求必须大于 1 */
    if (nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 检查第一个输入参数的类型，要求必须为双精度类型 */
    if (! mxIsDouble(prhs[0]))
    {
```

```
        mexErrMsgTxt("First input argument must be a double.");
    }

    /* 检查第二个输入参数的类型,要求必须为双精度的实数类型 */
    if (! mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]))
    {
        mexErrMsgTxt("Second input argument must be a real double.");
    }

    /* 获得第一个输入参数的维数,即用户输入的双精度类型阵列的维数 */
    nsubs=mxGetNumberOfDimensions(prhs[0]);

    /* 检查第二个输入参数的元素个数,要求其元素个数必须为 */
    if (mxGetNumberOfElements(prhs[1]) ! = nsubs)
    {
        mexErrMsgTxt("You must specify an index for each dimension.");
    }

    /* 动态分配内存 */
    subs=mxCalloc(nsubs,sizeof(int));

    /* 获得第一个输入参数实数部分数据的指针 */
    temp=mxGetPr(prhs[1]);

    /* 进行指定元素的标识转换,因为在 MATLAB 中,阵列以 1 为起始计数值,
        而在 C 语言中,却是以 0 为起始计数值,例如在 MATLAB 中阵列的元素
        (3,4),在 C 语言数组中表示为(2,3) */
    for (x=0;x<nsubs;x++)
    {
        subs[x]=(int)temp[x]-1;
        if (temp[x]> ((mxGetDimensions(prhs[0]))[x]) )
        {
            mxFree(subs);
            mexErrMsgTxt("You indexed above the size of the array.");
        }
    }
    /* 通过函数 mxCalcSingleSubscript 获得指定元素的索引值 */
    index = mxCalcSingleSubscript(prhs[0], nsubs, subs);

    /* 创建一个输出双精度类型的阵列,是复数或实数由第一个输入参数决定
        */
    plhs[0] = mxCreateDoubleMatrix(1, 1, mxIsComplex(prhs[0]) ?
                                         mxCOMPLEX : mxREAL);

    /* 释放动态分配的内存 */
```

```
        mxFree(subs);

        /* 为输出阵列赋值 */
        mxGetPr(plhs[0])[0]= mxGetPr(prhs[0])[index];
        if (mxIsComplex(prhs[0]))
        {
            mxGetPi(plhs[0])[0]= mxGetPi(prhs[0])[index];
        }
    }
```

对该程序编译后，在 MATLAB 命令提示符下键入如下命令，将显示以下的结果：

```
? b=rand(5)
b =
    0.2028    0.0153    0.4186    0.8381    0.5028
    0.1987    0.7468    0.8462    0.0196    0.7095
    0.6038    0.4451    0.5252    0.6813    0.4289
    0.2722    0.9318    0.2026    0.3795    0.3046
    0.1988    0.4660    0.6721    0.8318    0.1897
? c=[4,5];
? a=mxcalcsinglesubscript(b,c)
a =
0.3046
```

5. mxCalloc

功　能：动态内存分配。

语　法：

```
#include "matrix.h"
#include <stdlib.h>
void * mxCalloc(size_t n, size_t size);
```

说　明：函数 mxCalloc 是 MATLAB 提供的一个动态内存分配的函数，其功能与 C 语言中的标准库函数 calloc 的功能类似，但是一般在 MATLAB 的应用程序中，均使用函数 mxCalloc 来完成内存的分配工作，具体的原因，读者阅读了后续的内容自然会明白。这里必须注意的一点是，在 MEX 文件和独立可执行的 MATLAB 应用程序两种不同类型的接口程序中，函数 mxCalloc 所完成的动作是不同的：

- 在 MEX 文件中，函数 mxCalloc 将自动完成三个方面的工作：
  - 分配足够的堆空间；
  - 将所有的元素初始化为零；
  - 将返回的堆空间注册到 MATLAB 的自动内存管理机制中。在 MATLAB 的自动内存管理机制中，包含所有的由函数 mxCalloc 分配的内存，在程序结束时 MATLAB 自动内存管理机制将自动释放这些内存，从而避免内存泄漏。

- 在独立可执行的 MATLAB 应用程序中，使用函数 mxCalloc 时，函数将在内部自动调用 C 语言的标准库函数 calloc 来作为默认动作，完成内存分配。如果默认的动作执行失败，用户可以自定义一个内存分配函数，然后通过函数 mxSetAllocFcns 注册，这样在调用函数 mxCalloc 时，函数 mxCalloc 将默认地调用用户自定义的函数完成内存分配任务。

在默认情况下，在 MEX 文件中使用函数 mxCalloc 分配的内存为非持久内存，在程序结束时将被自动释放，这是通过 MATLAB 的自动内存管理机制完成的。但是如果用户不希望程序结束时内存被释放，可以通过使用函数 mexMakeMemoryPersistent 将函数 mxCalloc 分配的内存变为持久内存，这样在程序结束时，这部分内存也不会被释放，不过必须注意的是，在定义了持久内存之后，必须使用函数 mexAtExit 注册一个退出函数，用以在 MEX 文件被清除时，释放该持久内存，否则将导致内存泄漏。

举 例：程序 mxmalloc.c 是 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中。该程序的功能相当简单，它将一个输入的字符串类型的阵列中的内容拷贝到一个 C 语言风格的以空字符结尾的字符串中，其源代码如下：

```
/* mex.h 头文件包含，对于 MEX 文件，这是必不可少的 */
#include "mex.h"

/* 入口点函数 */
void
mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    char *buf;
    int    buflen;
    int    status;

    /* 检查输入参数和输出参数的个数 */
    if (nrhs != 1)
    {
        mexErrMsgTxt("One input argument required.");
    }

    if (nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 检查输入参数的类型 */
    if (!mxIsChar(prhs[0]) || (mxGetM(prhs[0]) != 1 ))
```

```
        {
            mexErrMsgTxt("Input argument must be a string.");
        }

        /* 获取输入字符串的大小,并分配足够的内存存储转换后的字符串。这里必
           须非常注意的一点是,在 MATLAB 中,字符串的存储按照 unicode,即 16
           位 ASCII,占用两个字节。*/

        buflen = mxGetN(prhs[0]) * sizeof(mxChar)+1;
        buf = mxMalloc(buflen);

        /* 拷贝字符串到内存 */
        status = mxGetString(prhs[0], buf, buflen);
        mexPrintf("The input string is: %s\n", buf);

        /* 释放内存 */
        mxFree(buf);

    }
```

6. mxChar

功　能:字符串类型 mxArray 结构体对象用于存储其元素的数据类型。

定　义:typedef Uint16 mxChar;

说　明:所有的字符串类型的 mxArray 结构体对象中的字符元素均声明为 mxChar 类型,在 MATLAB API 中 mxChar 被定义为一个无符号的 16 位整型数。

举　例:参见函数 mxCalloc 的范例程序 mxcalloc. c。

7. mxClassID

功　能:枚举类型数据,用于标识 mxArray 结构体的类型。

定　义:typedef enum

```
    {
        mxCELL_CLASS = 1,
        mxSTRUCT_CLASS,
        mxOBJECT_CLASS,
        mxCHAR_CLASS,
        mxSPARSE_CLASS,
        mxDOUBLE_CLASS,
        mxSINGLE_CLASS,
        mxINT8_CLASS,
        mxUINT8_CLASS,
        mxINT16_CLASS,
        mxUINT16_CLASS,
        mxINT32_CLASS,
```

```
    mxUINT32_CLASS,
    mxINT64_CLASS, /* place holder - future enhancements */
    mxUINT64_CLASS, /* place holder - future enhancements */
    mxUNKNOWN_CLASS = -1
} mxClassID;
```

说　明：mxClassID 定义了许多常数，用于标识 mxArray 结构体的类型，这些常数的含义分别如下：

- mxCELL_CLASS 代表单元类型的 mxArray 对象
- mxSTRUCT_CLASS 代表结构体类型的 mxArray 对象
- mxOBJECT_CLASS 代表对象类型的 mxArray 对象
- mxCHAR_CLASS 代表字符串类型的 mxArray 对象
- mxSPARSE_CLASS 代表稀疏类型的 mxArray 对象
- mxDOUBLE_CLASS 代表双精度类型的 mxArray 对象
- mxSINGLE_CLASS 代表单精度类型的 mxArray 对象
- mxINT8_CLASS 代表 int8 类型的 mxArray 对象
- mxUINT8_CLASS 代表 uint8 类型的 mxArray 对象
- mxINT16_CLASS 代表 int16 类型的 mxArray 对象
- mxUINT16_CLASS 代表 uint16 类型的 mxArray 对象
- mxINT32_CLASS 代表 int32 类型的 mxArray 对象
- mxUINT32_CLASS 代表 uint32 类型的 mxArray 对象
- mxINT64_CLASS 目前没有意义，为以后的发展预留
- mxUINT64_CLASS 目前没有意义，为以后的发展预留
- mxUNKNOWN_CLASS 代表类型不可确定的 mxArray 对象

举　例：程序 explore.c 是 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MEX\

中。该程序中包含了若干个的子函数，完成的主要功能为对所有的输入参数进行分析，并输出结果。该程序内容较多，我们将在程序中对各语句的功能进行说明。程序的源代码如下：

```
/* 各种头文件包含，其中对 mex.h 头文件的包含，对于 MEX 文件来说是必不可少
   的 */
#include <stdio.h>
#include <string.h>
#include "mex.h"

void        display_subscript(const mxArray *array_ptr, int index);
void        get_characteristics(const mxArray *array_ptr);
mxClassID   analyze_class(const mxArray *array_ptr);

/* analyze_cell 是程序对 MATLAB 单元阵列进行分析的一个子函数，其输入参数
   为一个指向单元阵列的指针。单元阵列是 MATLAB 中一种非常特殊的数据类
```

```
   型，这种类型阵列的每一个元素又为一个MATLAB阵列。通过子函数analyze_
   cell，用户可以对单元阵列中的子阵列进行访问和分析，并且显示相关的阵列信
   息 */
static void analyze_cell(const mxArray *cell_array_ptr)
{
  /* 变量定义 */
  int        total_num_of_cells; /* 用于存放单元阵列中元素的个数 */
  int        index;
  const mxArray *cell_element_ptr; /* 指向单元阵列中元素的指针 */

  /* 通过函数mxGetNumberOfElements获取单元阵列中元素的个数，并输出 */
  total_num_of_cells = mxGetNumberOfElements(cell_array_ptr);
  mexPrintf("total num of cells = %d\n", total_num_of_cells);
  mexPrintf("\n");

  /* 对单元阵列中的所有元素进行访问 */
  for (index=0; index<total_num_of_cells; index++)
  {
    mexPrintf("\n\n\t\tCell Element: ");
    display_subscript(cell_array_ptr, index);
    mexPrintf("\n");
    cell_element_ptr = mxGetCell(cell_array_ptr, index);
    if (cell_element_ptr == NULL)
      mexPrintf("\tEmpty Cell\n");
    else {
      get_characteristics(cell_element_ptr);
      analyze_class(cell_element_ptr);
      mexPrintf("\n");
        }
  }
  mexPrintf("\n");
}

/* 子函数analyze_structure的功能为对结构体阵列进行分析。结构体阵列是
   MATLAB中一种较为特殊的阵列，它的每一个阵列元素可以包含若干个域，每
   个域可以为不同类型的数据类型，与C语言中的结构体类似。子函数analyze_
   structure可以访问任何元素的任何域，并给出相应的信息。*/
static void analyze_structure(const mxArray *structure_array_ptr)
{
  /* 变量声明 */
  int   total_num_of_elements, number_of_fields;
  int   index, field_index;
  const char *field_name;
  const mxArray      *field_array_ptr;
```

```
  mexPrintf("\n");
  total_num_of_elements= mxGetNumberOfElements(structure_array_ptr);
  number_of_fields = mxGetNumberOfFields(structure_array_ptr);

  /* 访问结构体阵列的每一个元素 */
  for (index=0; index<total_num_of_elements; index++)
  {
    /* 对于给定的阵列元素,访问元素的每一个域 */
    for (field_index=0; field_index<number_of_fields; field_index++)
    {
      mexPrintf("\n\t\t");

      /* 根据索引值给出元素在阵列中的下标 */
      display_subscript(structure_array_ptr, index);

      /* 通过函数 mxGetFieldNameByNumber 根据索引值获取域的名字,并显
         示 */
      field_name = mxGetFieldNameByNumber(structure_array_ptr,
                                           field_index);
      mexPrintf(".%s\n", field_name);

      /* 通过函数 mxGetFieldByNumber 根据索引值获取域的指针 */
      field_array_ptr = mxGetFieldByNumber(structure_array_ptr,
                                  index, field_index);
      if (field_array_ptr == NULL)
        mexPrintf("\tEmpty Field\n");
      else
      {
        /* 对域进行分析 */
        get_characteristics(field_array_ptr);
        analyze_class(field_array_ptr);
        mexPrintf("\n");
      }
    }
      mexPrintf("\n\n");
    }
}

/* 子函数 analyze_string 的功能为对字符串阵列进行分析,并且一次输出字符串
   阵列的一行。在 MATLAB 中,字符串阵列的每一个元素为一个 mxChar 类型的
   字符,占用两个字节。*/
static void analyze_string(const mxArray *string_array_ptr)
{
  /* 变量声明 */
  char *buf;
```

```
int    number_of_dimensions;
const    int *dims;
int    buflen, d, page, total_number_of_pages, elements_per_page;

/* 分配足够的内存存放转换获得字符串 */
buflen = mxGetNumberOfElements(string_array_ptr) + 1;
buf = mxCalloc(buflen, sizeof(char));

/* 从输入的字符串中读取数据存放入缓存中 */
if (mxGetString(string_array_ptr, buf, buflen) != 0)
  mexErrMsgTxt("Could not convert string data.");

/* 获取输入字符串阵列的维数以及维数大小 */
dims = mxGetDimensions(string_array_ptr);
number_of_dimensions = mxGetNumberOfDimensions(string_array_ptr);

elements_per_page = dims[0] * dims[1];
/* total_number_of_pages = dims[2] x dims[3] x ... x dims[N-1] */
total_number_of_pages = 1;
for (d=2; d<number_of_dimensions; d++)
  total_number_of_pages *= dims[d];

/* 访问所有的字符,并按行输出 */
for (page=0; page < total_number_of_pages; page++)
{
  int row;
  for (row=0; row<dims[0]; row++)
  {
    int column;
    int index = (page * elements_per_page) + row;
    mexPrintf("\t");
    display_subscript(string_array_ptr, index);
    mexPrintf(" ");

    for (column=0; column<dims[1]; column++)
    {
      mexPrintf("%c",buf[index]);
      index += dims[0];
    }
    mexPrintf("\n");

  }
 }
}
```

```
/* 子函数 analyze_sparse 的功能为对输入的稀疏矩阵进行分析。稀疏矩阵是一种
   包含大量零元素的矩阵，在 MATLAB 中为了减小内存的消耗，为稀疏矩阵提供
   了与一般矩阵完全不同的存储方式，有关详细的存储方式，参见本数第一章。
   */
static void analyze_sparse(const mxArray *array_ptr)
{
  /* 变量声明 */
  double *pr, *pi;
  int    *ir, *jc;
  int    col, total=0;
  int    starting_row_index, stopping_row_index, current_row_index;
  int    n;

  /* 获取稀疏矩阵的所有数据指针 */
  pr = mxGetPr(array_ptr);
  pi = mxGetPi(array_ptr);
  ir = mxGetIr(array_ptr);
  jc = mxGetJc(array_ptr);

  /* 显示非零元素 */
  n = mxGetN(array_ptr);
  for (col=0; col<n; col++)
  {
    starting_row_index = jc[col];
    stopping_row_index = jc[col+1];
    if (starting_row_index == stopping_row_index)
      continue;
    else
    {
      for (current_row_index = starting_row_index;
      current_row_index < stopping_row_index;
      current_row_index++)
      {
        if (mxIsComplex(array_ptr))
        {
          mexPrintf("\t(%d,%d) = %g+%g i\n",
                    ir[current_row_index]+1,
                    col+1, pr[total], pi[total++]);
        }
        else
          mexPrintf("\t(%d,%d) = %g\n", ir[current_row_index]+1,
          col+1, pr[total++]);
      }
    }
  }
```

```
}

/* 子函数 analyze_int8 的功能为对 int8 类型的阵列进行分析 */
static void analyze_int8(const mxArray *array_ptr)
{
  signed char      *pr, *pi;
  char      total_num_of_elements, index;

  pr = (signed char *)mxGetPr(array_ptr);
  pi = (signed char *)mxGetPi(array_ptr);
  total_num_of_elements = mxGetNumberOfElements(array_ptr);

  for (index=0; index<total_num_of_elements; index++)
  {
    mexPrintf("\t");
    display_subscript(array_ptr, index);
    if (mxIsComplex(array_ptr))
      mexPrintf(" = %d + %di\n", *pr++, *pi++);
    else
      mexPrintf(" = %d\n", *pr++);
  }
}

/* 子函数 analyze_uint8 的功能为对 uint8 类型的阵列进行分析 */
static void analyze_uint8(const mxArray *array_ptr)
{
  unsigned char *pr, *pi;
  int      total_num_of_elements, index;

  pr = (unsigned char *)mxGetPr(array_ptr);
  pi = (unsigned char *)mxGetPi(array_ptr);
  total_num_of_elements = mxGetNumberOfElements(array_ptr);

  for (index=0; index<total_num_of_elements; index++)
  {
    mexPrintf("\t");
    display_subscript(array_ptr, index);
    if (mxIsComplex(array_ptr))
      mexPrintf(" = %u + %ui\n", *pr, *pi++);
    else
      mexPrintf(" = %u\n", *pr++);
  }
}

/* 子函数 analyze_int16 的功能为对 int16 类型的阵列进行分析 */
```

```
static void analyze_int16(const mxArray *array_ptr)
{
  short int        *pr, *pi;
  short int        total_num_of_elements, index;

  pr = (short int *)mxGetPr(array_ptr);
  pi = (short int *)mxGetPi(array_ptr);
  total_num_of_elements = mxGetNumberOfElements(array_ptr);

  for (index=0; index<total_num_of_elements; index++)
  {
    mexPrintf("\t");
    display_subscript(array_ptr, index);
    if (mxIsComplex(array_ptr))
      mexPrintf(" = %d + %di\n", *pr++, *pi++);
    else
      mexPrintf(" = %d\n", *pr++);
  }
}

/* 子函数 analyze_uint16 的功能为对 uint16 类型的阵列进行分析 */
static void analyze_uint16(const mxArray *array_ptr)
{
  unsigned short int        *pr, *pi;
  int                       total_num_of_elements, index;

  pr = (unsigned short int *)mxGetPr(array_ptr);
  pi = (unsigned short int *)mxGetPi(array_ptr);
  total_num_of_elements = mxGetNumberOfElements(array_ptr);

  for (index=0; index<total_num_of_elements; index++)
  {
    mexPrintf("\t");
    display_subscript(array_ptr, index);
    if (mxIsComplex(array_ptr))
      mexPrintf(" = %u + %ui\n", *pr++, *pi++);
    else
      mexPrintf(" = %u\n", *pr++);
  }
}

/* 子函数 analyze_int32 的功能为对 int32 类型的阵列进行分析 */
static void analyze_int32(const mxArray *array_ptr)
{
  int    *pr, *pi;
```

```
    int       total_num_of_elements, index;

    pr = (int *)mxGetPr(array_ptr);
    pi = (int *)mxGetPi(array_ptr);
    total_num_of_elements = mxGetNumberOfElements(array_ptr);

    for (index=0; index<total_num_of_elements; index++)
    {
      mexPrintf("\t");
      display_subscript(array_ptr, index);
      if (mxIsComplex(array_ptr))
        mexPrintf(" = %d + %di\n", *pr++, *pi++);
      else
        mexPrintf(" = %d\n", *pr++);
    }
}

/* 子函数 analyze_uint32 的功能为对 uint32 类型的阵列进行分析 */
static void analyze_uint32(const mxArray *array_ptr)
{
    unsigned int      *pr, *pi;
    int       total_num_of_elements, index;

    pr = (unsigned int *)mxGetPr(array_ptr);
    pi = (unsigned int *)mxGetPi(array_ptr);
    total_num_of_elements = mxGetNumberOfElements(array_ptr);

    for (index=0; index<total_num_of_elements; index++)
    {
      mexPrintf("\t");
      display_subscript(array_ptr, index);
      if (mxIsComplex(array_ptr))
        mexPrintf(" = %u + %ui\n", *pr++, *pi++);
      else
        mexPrintf(" = %u\n", *pr++);
    }
}

/* 子函数 analyze_single 的功能为对 single 类型的阵列进行分析 */
static void analyze_single(const mxArray *array_ptr)
{
    float *pr, *pi;
    int       total_num_of_elements, index;

    pr = (float *)mxGetPr(array_ptr);
```

```
    pi = (float *)mxGetPi(array_ptr);
    total_num_of_elements = mxGetNumberOfElements(array_ptr);

    for (index=0; index<total_num_of_elements; index++)
    {
      mexPrintf("\t");
      display_subscript(array_ptr, index);
      if (mxIsComplex(array_ptr))
        mexPrintf(" = %g + %gi\n", *pr++, *pi++);
      else
        mexPrintf(" = %g\n", *pr++);
    }
}

/* 子函数 analyze_double 的功能为对 double 类型的阵列进行分析 */
static void analyze_double(const mxArray *array_ptr)
{
    double *pr, *pi;
    int     total_num_of_elements, index;

    pr = (double *)mxGetPr(array_ptr);
    pi = (double *)mxGetPi(array_ptr);
    total_num_of_elements = mxGetNumberOfElements(array_ptr);

    for (index=0; index<total_num_of_elements; index++)
    {
      mexPrintf("\t");
      display_subscript(array_ptr, index);
      if (mxIsComplex(array_ptr))
        mexPrintf(" = %g + %gi\n", *pr++, *pi++);
      else
        mexPrintf(" = %g\n", *pr++);
    }
}

/* 子函数 analyze_full 的功能为对存储类型为满的阵列进行分析 */
static void analyze_full(const mxArray *numeric_array_ptr)
{
    mxClassID   category;

    category = mxGetClassID(numeric_array_ptr);
    switch (category)
    {
      case mxINT8_CLASS:
          analyze_int8(numeric_array_ptr);
```

```
            break;

        case mxUINT8_CLASS:
            analyze_uint8(numeric_array_ptr);
            break;

        case mxINT16_CLASS:
            analyze_int16(numeric_array_ptr);
            break;

        case mxUINT16_CLASS:
            analyze_uint16(numeric_array_ptr);
            break;

        case mxINT32_CLASS:
            analyze_int32(numeric_array_ptr);
            break;

        case mxUINT32_CLASS:
            analyze_uint32(numeric_array_ptr);
            break;

        case mxSINGLE_CLASS:
            analyze_single(numeric_array_ptr);
            break;

        case mxDOUBLE_CLASS:
            analyze_double(numeric_array_ptr);
            break;
    }
}

/* 子函数 display_subscript 的功能为根据索引值获取元素的下标 */
void display_subscript(const mxArray *array_ptr, int index)
{
    int       inner, subindex, total, d, q;
    int       number_of_dimensions;
    int       *subscript;
    const int *dims;

    number_of_dimensions = mxGetNumberOfDimensions(array_ptr);
    subscript = mxCalloc(number_of_dimensions, sizeof(int));
    dims = mxGetDimensions(array_ptr);

    mexPrintf("(");
```

```
    subindex = index;
    for (d = number_of_dimensions-1; d >= 0; d--)
    {
      for (total=1, inner=0; inner<d; inner++)
        total *= dims[inner];

      subscript[d] = subindex / total;
      subindex = subindex % total;
    }

    for (q=0; q<number_of_dimensions-1; q++)
      mexPrintf("%d,", subscript[q] + 1);
      mexPrintf("%d)", subscript[number_of_dimensions-1] + 1);

    mxFree(subscript);
}


/* 子函数 get_characteristics 的功能为获得输入参数的名字、维数和类型信息，并
   输出 */
void get_characteristics(const mxArray *array_ptr)
{
    /* 变量定义 */
    const char      *name; /* 存放输入参数的名字 */
    const char      *class_name; /* 存放输入参数的类型 */
    const int      *dims; /* 存放输入参数维数 */
    char *shape_string;
    char *temp_string;
    int     c;
    int     number_of_dimensions;
    int     length_of_shape_string;

    /* 显示 */
    mexPrintf("------------------------------------------------\n");

    /* 通过函数 mxGetName 获得输入阵列的名字并显示，如果该阵列没有定义名
       字，则输出 Unnamed */
    name = mxGetName(array_ptr);
    if (*name == '\0')
      mexPrintf("Name: Unnamed\n");
    else
      mexPrintf("Name: %s\n", name);

 /* 显示阵列的维数，例如 5×7×3。如果阵列的维数过大，无法按前面的格式显示，
    将直接输出维数，例如 12-D。*/
```

```
    /* 通过函数 mxGetNumberOfDimensions 获取阵列维数 */
    number_of_dimensions = mxGetNumberOfDimensions(array_ptr);

    /* 通过函数 mxGetDimensions 获得阵列每一维的大小 */
    dims = mxGetDimensions(array_ptr);

    /* 为格式输出分配内存,这样可以在其中添加符号 x */
    shape_string=(char *)mxCalloc(number_of_dimensions * 3,sizeof(char));
    shape_string[0]='\0';
    temp_string=(char *)mxCalloc(64, sizeof(char));

    for (c=0; c<number_of_dimensions; c++)
    {
      sprintf(temp_string, "%dx", dims[c]);
      strcat(shape_string, temp_string);
    }

    length_of_shape_string = strlen(shape_string);

    /* 将最后一个 x 替换为空格 */
    shape_string[length_of_shape_string-1]='\0';
    if (length_of_shape_string > 16)
      sprintf(shape_string, "%d-D\0", number_of_dimensions);

    /* 输出维数 */
    mexPrintf("Dimensions: %s\n", shape_string);

    /* 通过函数 mxGetClassName 获取阵列的类型并显示 */
    class_name = mxGetClassName(array_ptr);
    mexPrintf("Class Name: %s\n", class_name);

    /* 显示 */
    mexPrintf("------------------------------------------------\n");

    /* 释放内存 */
    mxFree(shape_string);
}

/* 子函数analyze_class 完成的功能为对单元阵列的元素进行类型分析,然后再调
   用相关的子函数对该元素进行分析 */
mxClassID analyze_class(const mxArray *array_ptr)
{

    /* 定义了一个 mxClassID 类型的变量 category,用以标识单元阵列元素的类型
       */
```

```
    mxClassID category;

    /* 通过函数 mxGetClassID 对元素的类型进行判断 */
    category = mxGetClassID(array_ptr);

    /* 根据元素的类型,选择相应的子函数进行处理 */
    switch (category)
    {
      /* 字符串类型 */
      case mxCHAR_CLASS:
          analyze_string(array_ptr);
          break;

      /* 结构类型 */
      case mxSTRUCT_CLASS:
          analyze_structure(array_ptr);
          break;

      /* 稀疏矩阵类型 */
      case mxSPARSE_CLASS:
          analyze_sparse(array_ptr);
          break;

      /* 单元阵列类型 */
      case mxCELL_CLASS:
          analyze_cell(array_ptr);
          break;

      /* 类型不详 */
      case mxUNKNOWN_CLASS:
          mexWarnMsgTxt("Unknown class.");
          break;

      /* 默认情况 */
      default:
          analyze_full(array_ptr);
          break;
    }

return(category);
}

/* MEX 文件入口点函数 */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
```

```
{
  int i;

/* 逐一地对输入参数进行分析 */
  for (i=0; i<nrhs; i++) {
    mexPrintf("\n\n");
    get_characteristics(prhs[i]);
    analyze_class(prhs[i]);
  }
}
```

对该程序编译后，在 MATLAB 下键入如下的命令，将得到以下结果：

```
? e=['afd';'ghj'];
? explore(e)
------------------------------------------------
Name: e
Dimensions: 2x3
Class Name: char
------------------------------------------------
    (1,1) afd
    (2,1) ghj
```

8. mxClearLogical

功　能：清除逻辑标志。

语　法：#include "matrix.h"

void mxClearLogical(mxArray *array_ptr);

说　明：通过函数 mxClearLogical 可以清除 mxArray 结构体对象的逻辑标志，该标志用来通知 MATLAB 将 mxArray 结构体数据作为数值数据处理，还是作为布尔数据处理。如果该标志为 on，那么 MATLAB 将数值 0 看作为逻辑假，而将非零的数值看作为逻辑真。此外用户还可以通过使用函数 mxSetLogical 为开启 mxArray 结构体对象的逻辑标志。

举　例：参见函数 mxIsLogical 的范例程序 mxislogical.c。

9. mxComplexity

功　能：用于判断阵列是否为复数类型。

定　义：typedef enum mxComplexity (mxREAL=0, mxCOMPLEX);

说　明：mxComplexity 为一个枚举类型的数据，它包含两个常量，其中 mxREAL 用来标识一个 mxArray 结构体对象为实数类型，不包含虚数部分，而 mxCOMPLEX 用来标识一个 mxArray 结构体对象为复数类型，包含虚数数据。在大量的 mx-函数中，均要使用到这两个常量。

举　例：参见函数 mxCalcSingleSubscript 的范例程序 mxcalcsinglesubscript.c。

10. mxCreateCellArray

功　能：创建一个 N 维的未赋值的单元阵列。

语　法：#include "matrix.h"

mxArray *mxCreateCellArray(int ndim, const int *dims);

说　明：通过函数 mxCreateCellArray，用户可以创建一个指定维数、指定大小的单元阵列，单元阵列的维数通过输入参数 ndim 来确定，而每一维的大小则通过数组 dims 来确定，其中 dims[0]用来指定阵列的行数，dims[1]用来指定阵列的列数，dims[2]用来指定阵列的页面数，后续的以此类推。如果用户希望创建一个 3 维的、大小为 3×4×5 的单元阵列，使用函数mxCreateCellArray 时，必须将参数设置为 ndims=3，dims[0]=3，dims[1]=4，dims[2]=5。如果函数执行成功将返回一个指向新创建的单元阵列的指针，并且将阵列的每一个单元均设置为空。如果函数执行不成功，在 MEX 文件中，整个程序将终止运行，并返回到 MATLAB 命令提示符下；而在独立可执行的 MATLAB 接口程序如引擎程序中，函数将返回一个 NULL 指针。一般来说，导致函数运行失败的原因主要有两种：

- 没有足够的空闲堆空间；
- 所确定的阵列的维数大于数组元素的个数。

举　例：下面是一段使用函数 mxCreateCellArray 的范例代码：

```
int ndims = 3;
int dims[] = {3,4,5};
mxArray *cell_array;
cell_array = mxCreateCellArray(ndims, dims);

/* 在独立可执行的 MATLAB 接口程序中，对函数执行的成功与否进行判断；而在
   MEX 文件中，则无需下面的代码对函数的执行成功与否进行判断，因为，在
   MEX 文件中，如果函数执行失败将直接返回 MATLAB 命令提示符 */
if (cell_array = NULL)
{
    printf("\n 创建单元阵列失败。");
    return;
}
```

11. mxCreateCellMatrix

功　能：创建一个二维的未赋值的单元阵列。

语　法：#include "matrix.h"

mxArray *mxCreateCellMatrix(int m, int n);

说　明：通过函数 mxCreateCellMatrix，用户可以创建一个指定行数和列数的二维单元阵列，具体的行数和列数由输入参数 m 和 n 确定。如果函数执行成功将返回一个指向新创建的单元阵列的指针，并且将阵列的每一个单元均设置为

空。如果函数执行不成功，在 MEX 文件中，整个程序将终止运行，并返回到 MATLAB 命令提示符下；而在独立可执行的 MATLAB 接口程序如引擎程序中，函数将返回一个 NULL 指针。一般来说，导致函数运行失败的原因主要有两种：

- 没有足够的空闲堆空间；
- 所确定的阵列的维数大于数组元素的个数。

总的来说，函数 mxCreateCellMatrix 完成的功能与函数 mxCreateCellArray 完成的功能类似，惟一的不同是函数 mxCreateCellMatrix 不能创建维数大于 2 的单元阵列。

举　例：程序 mxcreatecellmatrix. c 是 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中。该程序完成的功能相当简单，它将输入参数放入一个单元阵列中，然后使用 mex-函数 mexCallMATLAB 调用 MATLAB 命令显示单元阵列的内容，其源代码如下：

```
/* mex.h 头文件包含，这对于 MEX 文件来说是必不可少的 */
#include "mex.h"

/* 入口点函数 */
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    /* 变量声明 */
    mxArray *cell_array_ptr, *rhs[1];
    int i;

    /* 检查输入和输出变量的个数 */
    if (nrhs < 1)
    {
        mexErrMsgTxt("At least one input argument required.");
    }

    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 创建一个 nrhs X 1 的单元阵列 */
    cell_array_ptr = mxCreateCellMatrix(nrhs,1);

    /* 用输入参数填充单元阵列 */
    for( i=0; i<nrhs; i++)
    {
```

```
            mxSetCell(cell_array_ptr,i,mxDuplicateArray(prhs[i]));
        }

        rhs[0] = cell_array_ptr;

        /* 显示单元阵列的内容 */
        mexPrintf("\nThe contents of the created cell is:\n\n");
        mexCallMATLAB(0,NULL,1,rhs,"disp");
    }
```

对该程序进行编译后，在 MATLAB 命令提示符下键入如下命令，可以得到下面的结果：

```
? mxcreatecellmatrix(rand(3),'abcd')
The contents of the created cell is:
    [3x3 double]
    'abcd'
```

此外读者还可以参见 3.2.4 节的范例程序 uppercase.c。

12. mxCreateCharArray

功　能：创建一个 N 维的未赋值的字符串阵列。

语　法：#include "matrix.h"

mxArray *mxCreateCharArray(int ndim, const int *dims);

说　明：通过函数 mxCreateCharArray，用户可以创建一个指定维数的字符串阵列，该字符串阵列的维数通过输入参数 ndim 来确定，而每一维的大小则通过数组 dims 来确定，其中 dims[0]用来指定阵列的行数，dims[1]用来指定阵列的列数，dims[2]用来指定阵列的页面数，后续的以此类推。如果用户希望创建一个 3 维的、大小为 3×4×5 的字符串阵列，使用函数 mxCreateCharArray 时，必须将参数设置为 ndims=3,dims[0]=3,dims[1]=4,dims[2]=5。如果函数执行成功将返回一个指向新创建的字符串阵列的指针，但是并不为该字符串阵列赋值。如果函数执行不成功，在 MEX 文件中，整个程序将终止运行，并返回到 MATLAB 命令提示符下；而在独立可执行的 MATLAB 接口程序如引擎程序中，函数将返回一个 NULL 指针。导致函数运行失败的原因主要只有一种，即没有足够的空闲堆空间。

举　例：参见函数 mxCreateCharMatrixFromStrings 中的范例程序。

13. mxCreateCharMatrixFromStrings

功　能：创建一个二维的字符串阵列。

语　法：#include "matrix.h"

mxArray *mxCreateCharMatrixFromStrings(int m, const char **str);

说　明：通过函数 mxCreateCharMatrixFromStrings，用户可以创建一个 m 行的二维字符串阵列，并且通过输入参数中的字符串数组 str 对该阵列进行了赋值。

函数成功执行后，将返回一个字符串阵列的指针，并且所创建的字符串阵列的大小为 m×max，其中 max 为字符串数组中最长字符串的长度。如果函数执行不成功，在 MEX 文件中，整个程序将终止运行，并返回到 MATLAB 命令提示符下；而在独立可执行的 MATLAB 接口程序如引擎程序中，函数将返回一个 NULL 指针。

举　例：程序 mxcreatecharmatrixfromstr.c 是 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中。该程序的功能较为简单，它将输入的多个字符串阵列按先后顺序垂直连接为一个字符串阵列，其中每一行为原来的一个输入阵列。此外根据编译时的不同命令，在程序中对每一行字符串的长度作了相应的调整，添加空格或者空字符，使每一行的长度相同。程序的源代码如下：

```
/* mex.h 头文件包含，这对于 MEX 文件来说是必不可少的 */
#include "mex.h"

/* 入口点函数 */
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    int i;
    char *str[100];

    /* 检查输入参数和输出参数的个数 */
    if (nrhs < 2)
    {
        mexErrMsgTxt("At least two input arguments required.");
    }
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    for (i=0; i<nrhs; i++)
    {
        /* 检查输入参数的类型 */
        if(! mxIsChar(prhs[i]))
    {
        mexErrMsgTxt("Input must be of type char.");
    }

    /* 获取字符串阵列的所包含的字符串 */
    str[i] = mxArrayToString(prhs[i]);
    }
```

```
    /* 如果在编译时使用了参数 -D SPACE _ PADDING,程序中将使用空格填
       充字符串,否则使用空字符 NULL 来填充字符串 */
    #if defined (SPACE _ PADDING)
    {
        int dims[2];
        int i, j=0, m=nrhs, n=0;
        mxChar *charData;

        for (i = 0; i < m; i++)
        {
            int len;

            /* 获取最长的字符串的长度 */
            len = strlen(str[i]);
            if(len > n)
            {
                n = len;
            }
        }
        dims[0] = m;
        dims[1] = n;

        /* 创建字符串阵列 */
        plhs[0] = mxCreateCharArray(2, dims);
        charData = (mxChar *)mxGetData(plhs[0]);
        for (i = 0; i < m; i++)
        {
            j = 0;
            while (str[i][j] ! = '\0')
            {
                charData[j*m+i] = (mxChar) (unsigned char)str[i][j];
                j++;
            }
            while(j < n)
            {
                charData[j*m+i] = (mxChar) '';
                j++;
            }
        }
    }

#else

    plhs[0]= mxCreateCharMatrixFromStrings(nrhs, (const char **)str);
```

```
#endif

    /* 释放内存 */
    for (i=0; i<nrhs; i++)
    {
        mxFree(str[i]);
    }
}
```

对该程序编译后运行可以得到如下的结果：

```
? mxcreatecharmatrixfromstr('asd','fghq')
ans =
asd
fghq
```

14. mxCreateDoubleMatrix

功　能：创建一个二维的未赋值的双精度浮点类型的阵列。

语　法：#include "matrix.h"

mxArray * mxCreateDoubleMatrix (int m, int n, mxComplexity ComplexFlag);

说　明：通过函数 mxCreateDoubleMatrix，用户可以创建一个二维的双精度浮点类型的未赋值的阵列，该阵列的行数和列数可以通过输入参数 m 和 n 来确定；阵列为实数还是为复数，可以通过 mxComplexity 类型的输入参数 ComplexFlag 来确定。如果函数成功执行，将返回一个指向新创建阵列的指针；如果函数执行不成功，在 MEX 文件中，整个程序将终止运行，并返回到 MATLAB 命令提示符下；而在独立可执行的 MATLAB 接口程序如引擎程序中，函数将返回一个 NULL 指针。

举　例：在 3.2 节中的若干个范例程序中，均使用了该函数，请读者自行参阅。

15. mxCreateFull

说　明：该函数为一个过时的函数，其功能可以通过函数 mxCreateDoubleMatrix 来替代。如果需要使用已经存在对该函数调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

16. mxCreateNumericArray

功　能：创建一个 N 维的未赋值的数值阵列。

语　法：#include "matrix.h"

mxArray * mxCreateNumericArray(int ndim, const int * dims,
　　mxClassID class, mxComplexity ComplexFlag);

说　明：通过函数 mxCreateNumericArray，用户可以创建一个 N 维的未赋值的数值

阵列。阵列的维数可以通过输入参数 ndim 来确定；每一维的大小则通过输入的数组 dims 来确定，其中 dims[0]用来指定阵列的行数，dims[1]用来指定阵列的列数，dims[2]用来指定阵列的页面数，后续的以此类推；阵列的数值类型可以通过 mxClassID 类型的输入参数 class 来确定；实数或复数可以通过 mxComplexity 类型的输入参数 ComplexFlag 确定。如果用户希望创建一个 3 维的、大小为 3×4×5 的、mxINT16_CLASS 类型的实数阵列，使用函数 mxCreateNumericArray 时，必须将输入参数设置为如下形式：ndims = 3,dims[0] = 3,dims[1] = 4,dims[2] = 5,class = mxINT16_CLASS,ComplexFlag = 0。如果函数成功执行，将返回一个指向新创建阵列的指针，同时将该阵列的所有数据置为零；如果函数执行不成功，在 MEX 文件中，整个程序将终止运行，并返回到 MATLAB 命令提示符下；而在独立可执行的 MATLAB 接口程序如引擎程序中，函数将返回一个 NULL 指针。这里必须非常注意的一点是，函数 mxCreateNumericArray 将为所创建的数值阵列动态分配内存，在使用完该数值阵列后，应该使用函数 mxDestroyArray 对内存进行释放。

举　例：参见 3.2.5 节中的范例程序。

17. mxCreateSparse

功　能：创建一个未赋值的稀疏矩阵。

语　法：

```
#include "matrix.h"
mxArray *mxCreateSparse(int m, int n, int nzmax,
                        mxComplexity ComplexFlag);
```

说　明：通过函数 mxCreateSparse，用户可以创建一个二维的未赋值的稀疏矩阵。函数的四个输入参数的含义分别如下：

- m 为一个整型数，用以确定所创建的稀疏矩阵的行数；
- n 为一个整型数，用以确定所创建的稀疏矩阵的列数；
- nzmax 为一个整型数，用以确定稀疏矩阵中非零元素的最大个数；
- ComplexFlag 为一个 mxComplexity 类型的变量，用以确定稀疏矩阵为实数类型还是为复数类型。

如果函数执行成功，将返回一个指向未赋值的稀疏矩阵的指针，否则将返回一个空指针 NULL。这里必须说明的一点是函数所返回的稀疏矩阵并没有包含任何的数据信息，该矩阵不能被作为参数传递给任何的 MATLAB 稀疏矩阵处理函数使用。如果希望该稀疏矩阵有效，必须对 pr、ir、pi 和 jc 四个数组进行初始化。此外当对所返回的稀疏矩阵使用完毕后，必须使用函数 mxDestroyArray 对函数 mxCreateSparse 分配的内存进行释放。

举　例：参见 3.2.7 节的范例程序。

18. mxCreateString

功　能：创建一个 1×n 的字符串阵列。

语　法：#include "matrix.h"

mxArray *mxCreateString(const char *str);

说　明：通过函数 mxCreateString，用户可以创建一个 1×n 的字符串阵列，并且通过字符串 str 进行初始化，其中 n 为输入的字符串 str 的长度。

举　例：参见 3.2.1 节的范例程序。

19. mxCreateStructArray

功　能：创建一个 N 维的未赋值的结构体阵列。

语　法：#include "matrix.h"

mxArray *mxCreateStructArray(int ndim, const int *dims, int nfields,
const char **field_names);

说　明：通过函数 mxCreateStructArray，用户可以创建一个 N 维的未赋值的结构体阵列，其四个输入参数的含义分别如下：

- ndim 为一个整型数，用以确定结构体阵列的维数；
- dims 为一个整型数组，其数组元素分别用来确定结构体阵列每一维的大小，其中 dims[0]为结构体阵列的行数，dims[1]为结构体阵列的列数，dims[2]为结构体阵列的页面数，后续的以此类推；如果用户希望创建一个 3×4×5 的结构体阵列，则必须将参数 dims 设置为如下形式：dims[0] = 3, dims[1] = 4, dims[2] = 5；
- nfields 为一个整型数，用以确定结构体所包含的域的数量；
- field_names 为一个字符串数组，用以确定结构体各个域的名字。

如果函数执行成功，将返回一个结构体阵列的指针，否则函数返回一个空指针 NULL。结构体阵列的每一个元素的任何一个域均包含一个阵列的指针，函数在创建结构体阵列时，会将这些指针全部初始化为 NULL。当使用完函数返回的结构体阵列后，必须使用函数 mxDestroyArray 对所占用的内存进行释放。

举　例：程序 mxcreatestructarray.c 是 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中。该程序演示了如何将一个 C 语言的结构体转换为 MATLAB 的结构体阵列，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"
#include <string.h>

/* 宏定义 */
#define NUMBER_OF_STRUCTS
        (sizeof(friends)/sizeof(struct phonebook))
#define NUMBER_OF_FIELDS
        (sizeof(field_names)/sizeof(*field_names))
```

```
/* C语言结构体 */
struct phonebook
{
    const char *name;
    double phone;
};

/* 入口点函数 */
void
mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    /* 变量声明 */
    const char *field_names[] = {"name", "phone"};
    struct phonebook friends[] = {{"Jordan Robert", 3386},
                {"Mary Smith",3912},
                 {"Stacy Flora", 3238},{"Harry Alpert",3077}};
    int dims[2] = {1, NUMBER_OF_STRUCTS };
    int i, name_field, phone_field;

    /* 检查输入和输出变量的个数 */
    if (nrhs ! =0)
    {
        mexErrMsgTxt("No input argument required.");
    }
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 创建一个1×n的结构体阵列 */
    plhs[0] = mxCreateStructArray(2, dims, NUMBER_OF_FIELDS,
                                 field_names);

    name_field = mxGetFieldNumber(plhs[0],"name");
    phone_field = mxGetFieldNumber(plhs[0],"phone");

    /* 赋值 */
    for (i=0; i<NUMBER_OF_STRUCTS; i++)
    {
        mxArray *field_value;
        mxSetFieldByNumber(plhs[0], i, name_field,
                           mxCreateString(friends[i].name));
        field_value = mxCreateDoubleMatrix(1,1,mxREAL);
        *mxGetPr(field_value) = friends[i].phone;
```

```
                mxSetFieldByNumber(plhs[0],i,phone_field,field_value);
            }
        }
```

20. mxCreateStructMatrix

功　能:创建一个未赋值的二维结构体阵列。

语　法:#include "matrix.h"

```
mxArray *mxCreateStructMatrix(int m, int n, int nfields,
                              const char **field_names);
```

说　明:通过函数 mxCreateStructMatrix,用户可以创建一个未赋值的二维结构体阵列。函数的各输入参数的含义分别如下:

- m 为一个整型数,用以确定结构体阵列的行数;
- n 为一个整型数,用以确定结构体阵列的列数;
- nfields 为一个整型数,用以确定结构体所包含的域的数量;
- field_names 为一个字符串数组,用以确定结构体各个域的名字。

如果函数执行成功,将返回一个指向结构体阵列的指针,否则将返回一个空指针 NULL。

举　例:参见 3.2.3 节的范例程序。

21. mxDestroyArray

功　能:释放由 mxCreate 系列函数动态分配的内存。

语　法:#include "matrix.h"

```
void mxDestroyArray(mxArray *array_ptr);
```

说　明:通过函数 mxDestroyArray,用户可以释放由一个指定的阵列占用的内存,该内存不但包括阵列的一些特征域,而且包括与阵列相联系的一些数组,如 pr、pi、ir 和 jc。

举　例:参见函数 mexCallMATLAB 的范例程序 mexcallmatlab.c。

22. mxDuplicateArray

功　能:对一个阵列进行复制。

语　法:#include "matrix.h"

```
mxArray *mxDuplicateArray(const mxArray *in);
```

说　明:通过函数 mxDuplicateArray,用户可以对一个阵列进行完全的拷贝,即对输入的阵列 in 的所有内容进行复制。如果函数执行成功,函数将返回一个指向拷贝的指针。

举　例:参见 3.2.4 节的范例程序。

23. mxFree

功　能:释放由函数 mxCalloc 动态分配的内存。

语　法：#include "matrix.h"

void mxFree(void * ptr);

说　明：通过函数 mxFree，用户可以对由函数 mxCalloc 动态分配的内存进行释放。在所有的 MATLAB 接口应用程序，包括本章所讲述的 MEX 文件，以及后续的 MAT 文件应用程序和 MATLAB 引擎程序中，对堆内存的释放操作，都应该使用函数 mxFree 来代替标准的 C 语言的库函数 free。在 MEX 文件中，函数 mxFree 将自动完成两个方面工作：

- 调用 C 语言的库函数 free 释放以指针 ptr 为起始位置的连续的堆内存；
- 删除该段内存在 MATLAB 内存自动管理机制中注册的信息包。

MATLAB 的自动内存管理机制是 MATLAB 为了避免内存泄漏而提供的一种自动的内存管理机制，在它之中，保留了由函数 mxCalloc 和 mxCreate 系列函数分配的所有内存的信息包。当一个 MEX 文件执行完毕后，自动内存管理机制将释放所有与 MEX 文件相关的内存，并删除对应的信息包。通过内存的自动管理机制，用户无需手动地使用函数 mxFree 对内存进行释放。不过在对一段内存使用完毕后，立即进行释放，是一种良好的编程习惯。在默认情况下，由函数 mxCalloc 分配的内存为非持久的，但是如果通过函数 mexMakeMemoryPersistent 进行持久化后，这片内存就无法由 MATLAB 的自动内存管理机制来释放了，而必须通过使用 mexAtExit 来注册一个退出函数，在退出函数中使用 mxFree 进行内存释放。

举　例：参见函数 mxCalcSingleSubscript 的范例程序 mxcalcsinglesubscript.c。

24. mxFreeMatrix

说　明：该函数为一个过时的函数，其功能可以通过函数 mxDestroyArray 来替代。如果需要使用已经存在对该函数调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

25. mxGetCell

功　能：获得单元阵列中一个指定的单元。

语　法：#include "matrix.h"

mxArray * mxGetCell(const mxArray * array_ptr, int index);

说　明：通过函数 mxGetCell，用户可以获得单元阵列中一个指定的单元。函数的输入参数的含义分别如下：

- array_ptr 为一个指向单元阵列的指针；
- index 为希望提取的单元阵列单元的索引值，该索引值可以通过函数 mxCalcSingleSubscript 从希望提取单元的下标获取。

如果函数执行成功，将返回一个指向指定单元中阵列的指针，否则返回 NULL，表示执行失败，导致失败的原因可能有以下几种：

- 指定的单元并未被赋值；
- 确定的指针 array _ ptr 不是一个单元指针；
- 指定的索引值超出范围；
- 没有足够的内存空间存储返回的阵列。

举 例：参见 mxClassID 的范例程序 explore. c。

26. mxGetClassID

功 能：获取一个阵列的类型。

语 法：#include "matrix. h"

mxClassID mxGetClassID(const mxArray *array _ ptr);

说 明：通过函数 mxGetClassID，用户可以获得一个指定的阵列的类型，其输入参数为一个阵列指针，返回值为一个 mxClassID 类型的值。

举 例：参见 mxClassID 的范例程序 explore. c。

27. mxGetData

功 能：获得阵列的数据指针。

语 法：#include "matrix. h"

void *mxGetData(const mxArray *array _ ptr);

说 明：通过函数 mxGetData，用户可以获得由指针 array _ ptr 指向的阵列的数据指针。它不同于 mxGetPr 等函数的地方在于 mxGetData 的返回值为 void 类型的指针。

举 例：参见函数 mxCreateCharMatrixFromStrings 的范例程序。

28. mxGetDimensions

功 能：获得一个指向阵列维数大小的数组的指针。

语 法：#include "matrix. h"

const int *mxGetDimensions(const mxArray *array _ ptr);

说 明：通过函数 mxGetDimensions，用户可以获得一个指向阵列维数大小的数组的指针。该指针指向的数组的元素包含了阵列每一维的大小，其中数组的第一个元素为阵列的行数，数组的第二个元素为阵列的列数，第三个元素为阵列的页面数，后续的以此类推。函数的输入参数 array _ ptr 为一个指向某个阵列的指针。

举 例：参见函数 mxCreateCharMatrixFromStrings 的范例程序。

29. mxGetElementSize

功 能：获得存储阵列中元素所需的字节数量。

语 法：#include "matrix. h"

int mxGetElementSize(const mxArray *array _ ptr);

说 明：通过函数 mxGetElementSize，用户可以获得存储阵列中元素所需的字节数

量，其输入参数 array _ ptr 为指向某个阵列的指针。如果函数执行成功，将返回一个整数，用以表示存储阵列元素所需的字节的数量，如果执行失败，将返回 0，导致函数失败的原因一般为对阵列的类型不清。对于单元阵列和结构体阵列，函数将存放返回指向单元阵列元素和结构体阵列元素指针的字节数。

举　例：参见 3.2.5 节范例程序。

30. mxGetEps

功　能：获取 eps 的值。

语　法：#include "matrix.h"

double mxGetEps(void);

说　明：通过函数 mxGetEps，用户可以获得 MATLAB 的内部变量 eps 的值。

举　例：程序 mxgeteps.c 为 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中，它演示了函数 mxGetEps 的使用，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"
#include <math.h>

/* 入口点函数 */
void
mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    /* 变量声明 */
    const int *dims_first, *dims_second;
    int c, elements, j;
    double *first_ptr, *second_ptr, eps;

    /* 检查输入输出参数的个数 */
    if (nrhs != 2)
    {
        mexErrMsgTxt("Two input arguments required.");
    }
    if (nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 检查第一个输入参数的类型 */
    if (! mxIsDouble(prhs[0]) || ! mxIsDouble(prhs[1])
            || mxIsComplex(prhs[0]) || mxIsComplex(prhs[1]))
    {
```

```
        mexErrMsgTxt("Input arguments must be real of type double.");
    }

    /* 检查输入参数是否同维 */
    if ( mxGetNumberOfDimensions(prhs[0]) ! =
            mxGetNumberOfDimensions(prhs[1]))
    {
        mexErrMsgTxt("Inputs must have the same number
                of dimensions.\n");
    }

    dims_first = mxGetDimensions(prhs[0]);
    dims_second = mxGetDimensions(prhs[1]);

    /* 检查输入参数的每一维的大小是否相同 */
    for (c=0; c<mxGetNumberOfDimensions(prhs[0]); c++)
    {
        if (dims_first[c]! = dims_second[c])
        {
            mexErrMsgTxt("Inputs must have the same dimensions.\n");
        }
    }

    /* 获得输入参数的元素的个数 */
    elements=mxGetNumberOfElements(prhs[0]);

    /* 获得数据 */
    first_ptr = (double *)mxGetPr(prhs[0]);
    second_ptr = (double *)mxGetPr(prhs[1]);

    /* 构造输出 */
    plhs[0]=mxCreateDoubleMatrix(1,1,mxREAL);

    /* 获得EPS的值 */
    eps= mxGetEps( );

    /* 使用EPS作为容限,检测相等性 */
    for(j=0; j<elements; j++)
    {
        if((fabs(first_ptr[j] - second_ptr[j])) > (fabs(second_ptr[j] *
eps)))
        {
            break;
        }
    }
```

```
        if (j== elements)
        {
            mxGetPr(plhs[0])[0]=1.0;
        }
    }
```

31. mxGetField

功　能：获得指定结构体阵列指定元素的指定域的值。

语　法：#include "matrix.h"

```
mxArray *mxGetField(const mxArray *array_ptr, int index,
                    const char *field_name);
```

说　明：通过函数 mxGetField，用户可以获得指定结构体阵列指定元素的指定域的值，它的三个输入参数的含义分别如下：

- array_ptr 为一个指向某个结构体的指针；
- index 为希望提取的单元阵列单元的索引值，该索引值可以通过函数 mxCalcSingleSubscript 使用希望提取单元的下标获取；
- field_name 为域名。

如果函数执行成功，其返回值为一个阵列指针，用 C 语言的伪码返回结果可以表示为如下形式：

```
array_ptr[index].field_name
```

如果函数执行失败，将返回空指针。导致失败的原因一般有以下几种：

- 指针 array_ptr 并非一个指向结构体阵列的指针；
- index 的值超出结构体阵列的范围；
- field_name 为一个不存在的域名；
- 没有足够的内存空间存储返回的值。

举　例：参见 mxClassID 的范例程序 explore.c。

32. mxGetFieldByNumber

功　能：获得指定结构体阵列指定元素的指定域的值。

语　法：#include "matrix.h"

```
mxArray *mxGetFieldByNumber(const mxArray *array_ptr, int index,
                            int field_number);
```

说　明：函数 mxGetFieldByNumber 完成的功能与函数 mxGetField 完成的功能完全相同，它们之间惟一的不同是对域的确定方式的不同。在函数 mxGetField 中，通过直接输入域名字符串来确定域，而函数 mxGetFieldByNumber 则是通过域的索引来确定域。下面语句

```
field_num = mxGetFieldNumber(pa, "field_name");
mxGetFieldByNumber(pa, index, field_num);
```

的功能与语句

mxGetField(pa, index, "field_name");

的功能相同。

举 例：参见 mxClassID 的范例程序 explore.c。

33. mxGetFieldNameByNumber

功 能：通过索引获得域名。

语 法：#include "matrix.h"

```
const char *mxGetFieldNameByNumber(const mxArray *array_ptr,
                                   int field_number);
```

说 明：通过函数 mxGetFieldNameByNumber，用户可以获得指定域的名字，其输入参数的含义如下：

- array_ptr 为一个指向结构体阵列的指针；
- field_number 为域名的索引值，第一个域名的索引值为 0，第二个域名的索引值为 1，后续的以此类推。

如果函数执行成功，将返回一个指向字符串的指针，如果函数执行失败，将返回 NULL。一般情况下，导致函数出错的原因主要有以下两种：

- array_ptr 不是一个指向结构体阵列的指针；
- field_number 超出范围。

举 例：参见 mxClassID 的范例程序 explore.c。

34. mxGetFieldNumber

功 能：获得指定域名的索引值。

语 法：#include "matrix.h"

```
int mxGetFieldNumber(const mxArray *array_ptr, const char *field_
name);
```

说 明：函数 mxGetFieldNumber 的功能正好与函数 mxGetFieldNameByNumber 相反，它是通过给定域名来获得域名的索引值。其输入参数的含义分别如下：

- array_ptr 为一个指向结构体阵列的指针；
- field_name 为域名。

如果函数执行成功，将返回指定域名的索引值，如果函数执行不成功，将返回 -1。一般情况下，导致函数出错的原因主要有以下两种：

- array_ptr 不是一个指向结构体阵列的指针；
- field_name 不是结构体阵列所具有的域名。

举 例：参见函数 mxCreateStructArray 的范例程序。

35. mxGetImagData

功 能：获得阵列虚数部分数据的指针。

语 法：#include "matrix.h"

```
void *mxGetImagData(const mxArray *array_ptr);
```

说　明：函数 mxGetImagData 的功能与函数 mxGetPi 的功能相似，不过返回的指针类型为 void 类型。

举　例：参见函数 mxIsFinite 的范例程序。

36. mxGetInf

功　能：获得无穷大的值。

语　法：#include "matrix.h"

double mxGetInf(void);

说　明：通过函数 mxGetInf 用户可以获得 MATLAB 内部变量 inf 的值。inf 为一个常量，是 IEEE 所确定的用于代表正无穷大的符号。在一定的系统中，inf 的值是确定的，用户不能进行更改。在执行下列操作时，系统将返回 inf：

- 被 0 除；
- 数值溢出。

举　例：程序 mxgetinf.c 为 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中，它演示了函数 mxGetInf 的使用，其源代码如下：

```
/* 头文件包含 */
#include <limits.h>
#include "mex.h"
/* 入口点函数 */
void
mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    int i, n;
    double *pr, *pi;
    double inf, nan;

    /* 检查输入参数和输出参数的个数 */
    if (nrhs != 1)
    {
        mexErrMsgTxt("One input argument required.");
    }
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 检查输入参数的类型 */
    if (! mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]))
    {
        mexErrMsgTxt("Input argument must be of type real double.");
    }
```

```
        /* 复制输入参数 */
        plhs[0]=mxDuplicateArray(prhs[0]);

        pr = mxGetPr(plhs[0]);
        pi = mxGetPi(plhs[0]);
        n = mxGetNumberOfElements(plhs[0]);
        inf = mxGetInf( );
        nan=mxGetNaN( );

        /* 替换操作 */
        for(i=0; i < n; i++)
        {
            if (pr[i] == 0)
            {
                pr[i]=nan;
            }
            else if ( pr[i]>= INT_MAX)
            {
                pr[i]=inf;
            }
            else if (pr[i]<= INT_MIN)
            {
                pr[i]=-(inf);
            }
        }
    }
```

37. mxGetIr

功　能:获得稀疏矩阵的 ir 数组。

语　法:#include "matrix.h"

int *mxGetIr(const mxArray *array_ptr);

说　明:通过函数 mxGetIr,用户可以得到稀疏矩阵的 ir 数组,其输入参数为一个指向稀疏矩阵的阵列指针 array_ptr。如果函数执行成功,函数将返回一个指向 ir 数组第一个元素的整型指针。ir 数组元素的个数为稀疏矩阵中能够存放最多非零元素的个数,即 nzmax,并且 ir 数组中的每一个元素顺序地代表了稀疏矩阵中非零元素在稀疏矩阵中的行数。如果函数执行失败,将返回一个空指针 NULL。一般导致失败的原因主要有两点:

- array_ptr 所指向的阵列不是一个稀疏矩阵;
- array_ptr 为 NULL,这主要是由于前面调用函数 mxCreateSparse 失败导致。

举　例:参见 mxClassID 的范例程序 explore.c。

38. mxGetJc

功　能:获得稀疏矩阵的 jc 数组。

语　法:#include "matrix.h"

```
int *mxGetJc(const mxArray *array_ptr);
```

说　明:通过函数 mxGetJc,用户可以得到稀疏矩阵的 jc 数组,其输入参数为一个指向稀疏矩阵的阵列指针 array_ptr。如果函数执行成功,函数将返回一个指向 jc 数组第一个元素的整型指针。jc 数组的长度为 n+1,n 为稀疏矩阵的列数。

举　例:参见 mxClassID 的范例程序 explore.c。

39. mxGetM

功　能:获得阵列的行数。

语　法:#include "matrix.h"

```
int mxGetM(const mxArray *array_ptr);
```

说　明:通过函数 mxGetM,用户可以获得由输入参数 array_ptr 指定的阵列的行数,而不需要考虑阵列的实际维数。

举　例:参见 3.2.1 节范例程序。

40. mxGetN

功　能:获得阵列的列数。

语　法:#include "matrix.h"

```
int mxGetN(const mxArray *array_ptr);
```

说　明:通过函数 mxGetN,用户可以获得由输入参数 array_ptr 指定的阵列的列数,而不需要考虑阵列的实际维数。

举　例:参见 3.2.1 节范例程序。

41. mxGetName

功　能:获得阵列的名字。

语　法:#include "matrix.h"

```
const char *mxGetName(const mxArray *array_ptr);
```

说　明:通过函数 mxGetName,用户可以获得由输入参数 array_ptr 指定的阵列的名字。如果函数执行成功,函数将返回一个指向以 NULL 字符结尾的 C 语言字符串的指针。该字符串的长度是固定的,为 mxMAXNAM+1,而实际的阵列名字的长度则由空字符 NULL 来标识。常量 mxMAXNAM 是在头文件 mxArray.h 中定义。如果阵列没有名字,则返回的指针指向的字符串的第一个字符为\0。

举　例:参见 mxClassID 的范例程序 explore.c。

42. mxGetNaN

功 能：获得 NaN(not-a-number)的值。

语 法：
```
#include "matrix.h"
double mxGetNaN(void);
```

说 明：通过函数 mxGetNaN，用户可以获得系统内部变量 NaN 的值。NaN 为一个常量，是 IEEE 所确定的用于代表 not-a-number 的符号。在执行下列操作时，系统将返回 NaN：

- 0.0 / 0.0
- inf - inf

举 例：参见函数 mxGetInf 的范例程序 mxgetinf.c。

43. mxGetNumberOfDimensions

功 能：获得阵列的维数。

语 法：
```
#include "matrix.h"
int mxGetNumberOfDimensions(const mxArray *array_ptr);
```

说 明：通过函数 mxGetNumberOfDimensions，用户可以获得由输入参数 array_ptr 指定的阵列的维数。函数返回的最小值为 2。如果用户希望获得阵列每一维的大小，可以使用函数 mxGetDimensions。

举 例：参见 mxClassID 的范例程序 explore.c。

44. mxGetNumberOfElements

功 能：获得阵列中元素的个数。

语 法：
```
#include "matrix.h"
int mxGetNumberOfElements(const mxArray *array_ptr);
```

说 明：通过函数 mxGetNumberOfElements，用户可以获得由输入参数 array_ptr 指定的阵列元素的个数。与函数 mxGetClassID 结合使用，可以得到有关阵列的高级信息。

举 例：参见 mxClassID 的范例程序 explore.c。

45. mxGetNumberOfFields

功 能：获得结构体阵列的域的数量。

语 法：
```
#include "matrix.h"
int mxGetNumberOfFields(const mxArray *array_ptr);
```

说 明：通过函数 mxGetNumberOfFields，用户可以获得由输入参数 array_ptr 指定的结构体阵列的域的数量。一旦用户确切地知道了该值，就可以通过一个循环过程来访问结构体阵列的所有的域。

举 例：参见 mxClassID 的范例程序 explore.c。

46. mxGetNzmax

功　能：获得稀疏矩阵的三个数组 pr、pi 和 ir 的元素个数。

语　法：#include "matrix.h"

int mxGetNzmax(const mxArray *array_ptr);

说　明：通过函数 mxGetNzmax，用户可以获得稀疏矩阵中的 nzmax 域的值，稀疏矩阵通过该值来确定矩阵中可以包含的最大的非零元素的数量，同时该值也确定了稀疏矩阵的三个数组 pr、pi 和 ir 的元素个数。

举　例：程序 mxgetnzmax.c 为 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中，它演示了函数 mxGetNzmax 的使用，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"

/* 入口点函数 */
void
mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    int nzmax, nnz, columns;

    /* 检查输入参数和输出参数的个数 */
    if (nrhs != 1)
    {
        mexErrMsgTxt("One input argument required.");
    }
    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    if (! mxIsSparse(prhs[0]))
    {
        mexErrMsgTxt("Input argument must be a sparse array.");
    }
    nzmax = mxGetNzmax(prhs[0]);
    columns = mxGetN(prhs[0]);

    nnz = *(mxGetJc(prhs[0]) + columns);

    mexPrintf("Contains %d nonzero elements.\n", nnz);
    mexPrintf("Can store up to %d nonzero elements.\n", nzmax);
}
```

47. mxGetPi

功　能：获得阵列的虚数部分的数据指针。

语　法：#include "matrix.h"

double *mxGetPi(const mxArray *array_ptr);

说　明：通过函数 mxGetPi，用户可以得到由输入参数 array_ptr 指向的阵列的虚数部分的数据指针。如果函数执行成功，将返回一个双精度类型的指向包含虚数部分数据的数组的指针，否则返回 NULL。

举　例：参见 mxClassID 的范例程序 explore.c。

48. mxGetPr

功　能：获得阵列的实数部分的数据指针。

语　法：#include "matrix.h"

double *mxGetPr(const mxArray *array_ptr);

说　明：通过函数 mxGetPr，用户可以得到由输入参数 array_ptr 指向的阵列的实数部分的数据指针。如果函数执行成功，将返回一个双精度类型的指向包含实数部分数据的数组的指针，否则返回 NULL。

举　例：参见 mxClassID 的范例程序 explore.c。

49. mxGetScalar

功　能：获得某个阵列的实数部分的第一个数据。

语　法：#include "matrix.h"

double mxGetScalar(const mxArray *array_ptr);

说　明：通过函数 mxGetScalar，用户可以获得某个阵列的实数部分的第一个数据，其输入参数为一个指向某个阵列的指针。函数的返回值为一个双精度类型的数据，如果指针 array_ptr 指向的阵列的类型不为双精度类型，而为其他类型，函数会进行自动转换。如果指针 array_ptr 指向的阵列为单元阵列或者结构体阵列，函数的返回值为 0.0；如果指针 array_ptr 指向一个稀疏矩阵，则函数将返回稀疏矩阵中第一个非零元素的值；如果指针 array_ptr 指向一个空阵列，则返回值为一个不定值。一般函数 mxGetScalar 的使用对象为仅包含一个实数数据的阵列，对于含有多个元素的阵列或者多维阵列，函数永远返回阵列实数部分第一个数据的内容。

举　例：参见函数 mexGet 的范例程序 mexget.c。

50. mxGetString

功　能：获得字符串阵列的内容。

语　法：#include "matrix.h"

int mxGetString(const mxArray *array_ptr, char *buf, int buflen);

说　明：通过函数 mxGetString，用户可以获得字符串阵列所包含的字符串。函数的

三个输入参数的含义分别如下：

- array _ ptr 为一个指向字符串类型阵列的指针；
- buf 为存放读取的字符串的缓存的起始指针；
- buflen 为存放字符串的缓存的长度。

如果函数成功，返回值为 0，否则为 1。一般导致函数执行失败的原因主要有以下几点：

- array _ ptr 所指向的阵列不是字符串类型的阵列；
- buflen 所确定的缓存的长度小于字符串阵列包含的字符串的长度，这种情况下，函数将返回 1，并且将字符串截断。

缓存 buf 中的字符串为一个 C 语言风格的字符串，其最大长度为 buflen-1，并且以空字符 NULL 标识结束。如果参数 array _ ptr 指向的字符串阵列包含若干行，函数在执行时，会按列优先的顺序将它们一并读出，并且连接成为一个长的字符串。

举　例：参见 mxClassID 的范例程序 explore.c。

51. mxIsCell

功　能：判断阵列是否为单元阵列。

语　法：#include "matrix.h"

```
bool mxIsCell(const mxArray  * array _ ptr);
```

说　明：通过函数 mxIsCell，用户可以判断由输入参数 array _ ptr 所指向的阵列是否为单元阵列。如果返回值为 1，则说明 array _ ptr 所指向的阵列为单元阵列；如果返回值为 0，则说明 array _ ptr 所指向的阵列不是单元阵列。此外函数 mxIsCell 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array _ ptr) == mxCELL _ CLASS
```

关于函数 mxGetClassID 和常量 mxCELL _ CLASS 的说明，请读者参阅前面的内容。

举　例：参见 3.2.4 节的范例程序。

52. mxIsChar

功　能：判断阵列是否为字符串类型的阵列。

语　法：#include "matrix.h"

```
bool mxIsChar(const mxArray  * array _ ptr);
```

说　明：通过函数 mxIsChar，用户可以判断由输入参数 array _ ptr 所指向的阵列是否为字符串类型的阵列。如果返回值为 1，则说明 array _ ptr 所指向的阵列为字符串类型的阵列；如果返回值为 0，则说明 array _ ptr 所指向的阵列不是字符串类型的阵列。此外函数 mxIsChar 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array _ ptr) == mxCHAR _ CLASS
```

关于函数 mxGetClassID 和常量 mxCHAR _ CLASS 的说明,请读者参阅前面的内容。

举 例:参见 3.2.1 节的范例程序。

53. mxIsClass

功 能:判断阵列是否为指定类型的阵列。

语 法:#include "matrix.h"

bool mxIsClass(const mxArray *array _ ptr, const char *name);

说 明:通过函数 mxIsClass,用户可以判断一个阵列是否为指定类型的阵列。函数的两个输入参数的含义分别如下:

- array _ ptr 为指向某个阵列的指针;
- name 为一个字符串变量,用以确定阵列的类型,表 3.2 为各字符串常量与类型常量的对照表。

表 3.2 常量对照表

字符串常量	类型常量
"double"	mxDOUBLE _ CLASS
"sparse"	mxSPARSE _ CLASS
"cell"	mxCELL _ CLASS
"char"	mxCHAR _ CLASS
"struct"	mxSTRUCT _ CLASS
"single"	mxSINGLE _ CLASS
"int8"	mxINT8 _ CLASS
"unit8"	mxUINT8 _ CLASS
"int16"	mxINT16 _ CLASS
"uint16"	mxUINT16 _ CLASS
"int32"	mxINT32 _ CLASS
"uint32"	mxUINT32 _ CLASS
<class _ name>(自定义)	mxOBJECT _ CLASS
"unknown"	mxUNKNOWN _ CLASS

如果函数返回值为 TRUE,则说明阵列的确为指定的字符串。

举 例:程序 mxisclass.c 为 MATLAB 提供的一个范例程序,存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中,它演示了函数 mxIsClass 的使用,其源代码如下:

```
/* 头文件包含 */
#include "mex.h"

/* 入口点函数 */
void
mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    /* 变量说明 */
    mxArray *output, *input;
```

```
    int number_of_fields, field_num;

    /* 检查输入参数和输出参数的变量 */
    if (nrhs != 0)
    {
        mexErrMsgTxt("No input argument required.");
    }

    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }
    input=mxCreateString("sin(3*x)");

    /* 使用函数 mexCallMATLAB 调用 MATLAB 命令 */
    mexCallMATLAB(1,&output, 1, &input,"inline");
    mxDestroyArray(input);

    /* 判断阵列的类型 */
    if (! mxIsClass(output, "inline"))
    {
        mxDestroyArray(output);
        mexErrMsgTxt("Failed to create an object of class inline");
    }

    number_of_fields = mxGetNumberOfFields(output);
    mexPrintf("This object contains the following fields:\n");
    mexPrintf("name\t\tclass\t\tvalue\n");
    mexPrintf("--------------------------------------\n");

    /* 获得域名 */
    for (field_num=0; field_num<number_of_fields; field_num++)
    {
        mxArray *pa;
        mexPrintf("%s", mxGetFieldNameByNumber(output,
                                 field_num));
        pa = mxGetFieldByNumber(output, 0, field_num);
        mexPrintf("\t\t%s\t\t", mxGetClassName(pa));
        mexCallMATLAB(0, NULL, 1, &pa, "disp");
    }

    mxDestroyArray(output);
}
```

54. mxIsComplex

功　能：判断阵列是否为复数类型。

语　法：#include "matrix.h"

bool mxIsComplex(const mxArray *array_ptr);

说　明：通过函数 mxIsComplex，用户可以判断由输入参数 array_ptr 所指向的阵列是否为复数类型的阵列。如果返回值为 1，则说明 array_ptr 所指向的阵列为复数类型的阵列；如果返回值为 0，则说明 array_ptr 所指向的阵列不是复数类型的阵列。

举　例：参见函数 mxIsLogical 的范例程序 mxislogical.c。

55. mxIsDouble

功　能：判断阵列是否为双精度类型。

语　法：#include "matrix.h"

bool mxIsDouble(const mxArray *array_ptr);

说　明：通过函数 mxIsDouble，用户可以判断由输入参数 array_ptr 所指向的阵列是否为双精度类型的阵列。如果返回值为 1，则说明 array_ptr 所指向的阵列为双精度类型的阵列；如果返回值为 0，则说明 array_ptr 所指向的阵列不是双精度类型的阵列。

举　例：参见 3.2.3 节的范例程序。

56. mxIsEmpty

功　能：判断阵列是否为空。

语　法：#include "matrix.h"

bool mxIsEmpty(const mxArray *array_ptr);

说　明：通过函数 mxIsEmpty，用户可以判断由输入参数 array_ptr 所指向的阵列是否为空。如果返回值为 1，则说明 array_ptr 所指向的阵列为空；如果返回值为 0，则说明 array_ptr 所指向的阵列为非空。

举　例：参见函数 mxIsLogical 的范例程序 mxislogical.c。

57. mxIsFinite

功　能：判断一个值是否为有限值。

语　法：#include "matrix.h"

bool mxIsFinite(double value);

说　明：通过函数 mxIsFinite，用户可以判断由输入参数 value 所代表的双精度类型的浮点数是否为 inf 或者 NaN。如果函数的返回值为 1，说明输入的参数为有限值，不为 inf 和 NaN；如果返回 0，则说明输入参数必为 inf 和 NaN 两者之一。

举　例：程序 mxisfinite.c 为 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中，它演示了函数 mxIsFinite 的使用，其源代码如下：

```
/* 头文件包含 */
#include <limits.h>
#include "mex.h"

/* dtoi32 为一个子函数,其功能为在判断输入参数 d 是否为 inf 或 NaN 的前提上,
   进行不同的处理 */
static int dtoi32(double d)
{
    int i=0;

    if(mxIsFinite(d))
    {
        /* 如果 d 为有限值 */
        if(d < (double)INT_MAX && d > (double)INT_MIN)
        {
            /* 如果 d 可以用 32 位整数表示,则将 d 转换为 32 位整型数 */
            i = (int) d;
        }
        else
        {
            /* 如果 d 不可以用 32 位整数表示,判断 d 是过大还是过小 */
            i = ((d > 0) ? INT_MAX : INT_MIN);
        }
    }
    else if(mxIsInf(d))
    {
        /* 如果 d 为无穷大,判断 d 是过大还是过小 */
        i = ( (d > 0) ? INT_MAX : INT_MIN);
    }
    else if(mxIsNaN(d))
    {
        /* 如果 d 为 NaN,将 d 设置为零 */
        mexWarnMsgTxt("dtoi32: NaN detected. Translating to 0.\n");
        i = 0;
    }
    return i;
}

/* 入口点函数 */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
    /* 变量声明 */
    int i, n;
    double *pr, *pi;
```

```
int *pri32, *pii32;

/* 检查输入输出参数的个数 */
if (nrhs != 1)
{
    mexErrMsgTxt("One input argument required.");
}
if(nlhs > 1)
{
    mexErrMsgTxt("Too many output arguments.");
}

/* 检查输入参数的类型 */
if (! (mxIsDouble(prhs[0])))
{
    mexErrMsgTxt("Input argument must be of type double.");
}

/* 检查输入参数是否为空 */
if(mxIsEmpty(prhs[0]))
{
    mexWarnMsgTxt("Input argument is empty\n");
}

pr = mxGetPr(prhs[0]);
pi = mxGetPi(prhs[0]);
n = mxGetNumberOfElements(prhs[0]);

/* 创建 mxINT32 类型的数值阵列 */
plhs[0] = mxCreateNumericArray(
              mxGetNumberOfDimensions(prhs[0]),
              mxGetDimensions(prhs[0]),
              mxINT32_CLASS,
              (mxIsComplex(prhs[0]) ? mxCOMPLEX : mxREAL));
pri32 = mxGetData(plhs[0]);
pii32 = mxGetImagData(plhs[0]);

/* 将输入阵列的实数部分的每一个元素转化为 int32 */
for(i=0; i < n; i++)
{
    pri32[i] = dtoi32(pr[i]);
}

/* 如果输入阵列存在虚数部分,则将虚部数据也转化为 int32 */
if(pii32 != NULL)
```

```
            {
                bool empty_image_data = true;
                for(i=0; i < n; i++)
                {
                    pii32[i] = dtoi32(pi[i]);
                    if(pii32[i] != 0)
                    {
                        empty_image_data = false;
                    }
                }

                if (empty_image_data)
                {
                    mxFree(pii32);
                    mxSetImagData(plhs[0], NULL);
                }
            }
        }
```

58. mxIsFromGlobalWS

功　能：判断阵列是否是从 MATLAB 的全局工作空间中获得。

语　法：#include "matrix.h"

bool mxIsFromGlobalWS(const mxArray *array_ptr);

说　明：通过函数 mxIsFromGlobalWS，用户可以判断由输入参数 array_ptr 所指向的阵列是否为从 MATLAB 的全局工作空间中获得。如果函数的返回值为 1，说明输入参数 array_ptr 所指向的阵列是从 MATLAB 的全局工作空间中获得，否则返回 0。

举　例：参见 5.4.1 节函数 matGetArrayHeader 的范例程序。

59. mxIsFull

说　明：该函数为一个过时的函数，MATLAB V5.X 版本不对其提供支持。如果需要使用已经存在对该函数调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。在 MATLAB V5.X 版本的 MEX 文件中，用户可以使用语句

```
if(! mxIsSparse(prhs[0]))
```

来代替语句

```
if(mxIsFull(prhs[0]))
```

完成的功能。

60. mxIsInf

功　能：判断一个双精度类型的数值是否为 inf。

语　法：
```
#include "matrix.h"
bool mxIsInf(double value);
```

说　明：通过函数 mxIsInf，用户可以判断由输入参数 value 所代表的双精度类型的数值是否为 inf。如果函数返回值为真，说明输入参数 value 所代表的双精度类型的数值为 inf，否则函数返回 0。

举　例：参见函数 mxIsFinite 的范例程序 mxisfinite.c。

61. mxIsInt8

功　能：判断阵列的数据类型是否为 8 位的整数类型。

语　法：
```
#include "matrix.h"
bool mxIsInt8(const mxArray *array_ptr);
```

说　明：通过函数 mxIsInt8，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为 8 位的整数类型。如果返回值为 1，则说明 array_ptr 所指向的阵列为 8 位的整数类型；如果返回值为 0，则说明 array_ptr 所指向的阵列不是 8 位的整数类型。此外函数 mxIsInt8 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array_ptr) == mxINT8_CLASS
```

关于函数 mxGetClassID 和常量 mxINT8_CLASS 的说明，请读者参阅前面的内容。

62. mxIsInt16

功　能：判断阵列的数据类型是否为 16 位的整数类型。

语　法：
```
#include "matrix.h"
bool mxIsInt16(const mxArray *array_ptr);
```

说　明：通过函数 mxIsInt16，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为 16 位的整数类型。如果返回值为 1，则说明 array_ptr 所指向的阵列为 16 位的整数类型；如果返回值为 0，则说明 array_ptr 所指向的阵列不是 16 位的整数类型。此外函数 mxIsInt16 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array_ptr) == mxINT16_CLASS
```

关于函数 mxGetClassID 和常量 mxINT16_CLASS 的说明，请读者参阅前面的内容。

63. mxIsInt32

功　能：判断阵列的数据类型是否为 32 位的整数类型。

语　法：#include "matrix.h"

bool mxIsInt32(const mxArray *array_ptr);

说　明：通过函数 mxIsInt32，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为 32 位的整数类型。如果返回值为 1，则说明 array_ptr 所指向的阵列为 32 位的整数类型；如果返回值为 0，则说明 array_ptr 所指向的阵列不是 32 位的整数类型。此外函数 mxIsInt32 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array_ptr) == mxINT32_CLASS
```

关于函数 mxGetClassID 和常量 mxINT32_CLASS 的说明，请读者参阅前面的内容。

64. mxIsLogical

功　能：判断阵列的数据类型是否为逻辑类型。

语　法：#include "matrix.h"

bool mxIsLogical(const mxArray *array_ptr);

说　明：通过函数 mxIsLogical，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为逻辑类型。如果函数返回 1，说明 array_ptr 所指向的阵列的数据类型为逻辑类型，这时阵列中所有的非零元素将代表逻辑真，而零元素代表逻辑假；如果函数返回 0，则说明 array_ptr 所指向的阵列的数据类型不为逻辑类型。

举　例：程序 mxislogical.c 为 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中，它演示了函数 mxIsLogical 的使用，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"

/* 入口点函数 */
void
mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    /* 变量声明 */
    mxArray *array_ptr;
    char    *variable;

    /* 检查输入输出参数的个数 */
    if (nrhs != 1)
    {
        mexErrMsgTxt("One input argument required.");
    }
    if(nlhs > 1)
```

```
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 检查输入变量是否为字符串类型 */
    if (! (mxIsChar(prhs[0])))
    {
        mexErrMsgTxt("Input must be of type string.\n.");
    }

    /* 获取输入参数的内容 */
    variable= mxArrayToString(prhs[0]);
    array_ptr = (mxArray *)mexGetArrayPtr(variable, "caller");
    if (array_ptr == NULL)
    {
        mexErrMsgTxt("Could not get variable.\n");
    }

    if (mexIsGlobal(array_ptr))
    {
        mexErrMsgTxt("The variable you requested is global.\n");
    }
    mexPrintf("%s is not a global\n",variable);

    if(mxIsLogical(array_ptr))
    {
        mxClearLogical(array_ptr);
    }
    else
    {
        mxSetLogical(array_ptr);
    }
}
```

65. mxIsNaN

功　能:判断一个双精度类型的数值是否为 not-a-number。

语　法:#include "matrix.h"

bool mxIsNaN(double value);

说　明:通过函数 mxIsNaN,用户可以判断由输入值为参数 value 所代表的双精度类型的数值是否为 not-a-number。如果函数返回真,说明输入参数 value 所代表的双精度类型的数值为 not-a-number,否则函数返回 0。

举　例:参见函数 mxIsFinite 的范例程序 mxisfinite.c。

66. mxIsNumeric

功 能：判断阵列是否为数值阵列。

语 法：#include "matrix.h"

```
bool mxIsNumeric(const mxArray *array_ptr);
```

说 明：通过函数 mxIsNumeric，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为数值类型。如果函数返回 1，说明 array_ptr 所指向的阵列的数据类型为数值类型；如果函数返回 0，则说明 array_ptr 所指向的阵列的数据类型不为数值类型。当阵列为以下类型时，函数将返回 1：

- mxDOUBLE_CLASS
- mxDOUBLE_CLASS
- mxSPARSE_CLASS
- mxSINGLE_CLASS
- mxINT8_CLASS
- mxUINT8_CLASS
- mxINT16_CLASS
- mxUINT16_CLASS
- mxINT32_CLASS
- mxUINT32_CLASS

当阵列为以下类型时，函数将返回 0：

- mxCELL_CLASS
- mxCHAR_CLASS
- mxOBJECT_CLASS
- mxSTRUCT_CLASS
- mxUNKNOWN_CLASS

举 例：参见 3.2.8 节的范例程序。

67. mxIsSingle

功 能：判断阵列的数据类型是否为单精度的浮点类型。

语 法：#include "matrix.h"

```
bool mxIsSingle(const mxArray *array_ptr);
```

说 明：通过函数 mxIsSingle，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为单精度的浮点类型。如果返回值为 1，则说明 array_ptr 所指向的阵列为单精度的浮点类型；如果返回值为 0，则说明 array_ptr 所指向的阵列不是单精度的浮点类型。此外函数 mxIsSingle 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array_ptr) == mxSINGLE_CLASS
```

关于函数 mxGetClassID 和常量 mxSINGLE_CLASS 的说明，请读者参阅

前面的内容。

68. mxIsSparse

功　能：判断阵列是否为稀疏阵列。

语　法：#include "matrix.h"

bool mxIsSparse(const mxArray *array_ptr);

说　明：通过函数 mxIsSparse，用户可以判断由输入参数 array_ptr 所指向的阵列是否为稀疏阵列。如果返回值为 1，则说明 array_ptr 所指向的阵列为稀疏阵列；如果返回值为 0，则说明 array_ptr 所指向的阵列不是稀疏阵列。

举　例：参见 3.2.7 节的范例程序。

69. mxIsString

说　明：该函数为一个过时的函数，MATLAB V5.X 版本不对其提供支持。如果需要使用已经存在对该函数调用的 MEX 文件，而不希望对源程序进行修改，那么在对这类 MEX 文件进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MEX 文件。

70. mxIsStruct

功　能：判断阵列是否为结构体阵列。

语　法：#include "matrix.h"

bool mxIsStruct(const mxArray *array_ptr);

说　明：通过函数 mxIsStruct，用户可以判断由输入参数 array_ptr 所指向的阵列是否为结构体阵列。如果返回值为 1，则说明 array_ptr 所指向的阵列为结构体阵列；如果返回值为 0，则说明 array_ptr 所指向的阵列不是结构体阵列。

举　例：参见 3.2.3 节的范例程序。

71. mxIsUint8

功　能：判断阵列的数据类型是否为 8 位的无符号整数类型。

语　法：#include "matrix.h"

bool mxIsUint8(const mxArray *array_ptr);

说　明：通过函数 mxIsUint8，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为 8 位的无符号整数类型。如果返回值为 1，则说明 array_ptr 所指向的阵列为 8 位的无符号整数类型；如果返回值为 0，则说明 array_ptr 所指向的阵列不是 8 位的无符号整数类型。此外函数 mxIsInt8 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array_ptr) == mxUINT8_CLASS
```

关于函数 mxGetClassID 和常量 mxUINT8_CLASS 的说明，请读者参阅前面的内容。

72. mxIsUint16

功　能：判断阵列的数据类型是否为 16 位的无符号整数类型。

语　法：#include "matrix.h"

　　bool mxIsUint16(const mxArray *array_ptr);

说　明：通过函数 mxIsUint16，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为 16 位的无符号整数类型。如果返回值为 1，则说明 array_ptr 所指向的阵列为 16 位的无符号整数类型；如果返回值为 0，则说明 array_ptr 所指向的阵列不是 16 位的无符号整数类型。此外函数 mxIsInt16 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array_ptr) == mxUINT16_CLASS
```

关于函数 mxGetClassID 和常量 mxUINT16_CLASS 的说明，请读者参阅前面的内容。

73. mxIsUint32

功　能：判断阵列的数据类型是否为 32 位的无符号整数类型。

语　法：#include "matrix.h"

　　bool mxIsUint32(const mxArray *array_ptr);

说　明：通过函数 mxIsUint32，用户可以判断由输入参数 array_ptr 所指向的阵列的数据类型是否为 32 位的无符号整数类型。如果返回值为 1，则说明 array_ptr 所指向的阵列为 32 位的无符号整数类型；如果返回值为 0，则说明 array_ptr 所指向的阵列不是 32 位的无符号整数类型。此外函数 mxIsInt32 的功能还可以通过下面的语句来完成：

```
mxGetClassID(array_ptr) == mxUINT32_CLASS
```

关于函数 mxGetClassID 和常量 mxUINT32_CLASS 的说明，请读者参阅前面的内容。

74. mxMalloc

功　能：使用 MATLAB 的内存管理器进行动态内存分配。

语　法：#include "matrix.h"

　　#include <stdlib.h>

　　void *mxMalloc(size_t n);

说　明：通过函数 mxMalloc，用户可以方便地在 MATLAB 接口应用程序中进行动态内存分配，其输入参数 n 代表希望分配的内存的大小，单位为字节。如果函数执行成功，将返回一个 void 类型的指针，指向所分配内存区域的起始字节；如果函数执行失败，MEX 文件和其他的 MATLAB 接口应用程序的处理方法不同：在 MEX 文件中，函数将终止整个 MEX 文件的执行，返回到 MATLAB 的命令提示符下；而在其他的 MATLAB 接口应用程序中，函数

将返回一个空指针 NULL。造成不同返回结果的原因是在 MEX 文件和其他的 MATLAB 接口应用程序中，函数 mxMalloc 的工作方法不一致。在 MEX 文件中，函数 mxMalloc 将自动完成两方面的任务：

- 分配足够的连续的堆内存空间；
- 将所分配的内存空间在 MATLAB 的自动内存管理机制中注册，这样对于非持久的内存，在 MEX 文件结束执行时，MATLAB 会自动释放这些内存，而不用用户显式地调用内存释放函数。在默认情况下，内存分配函数分配的内存皆为非持久内存，当通过函数 mexMakeMemoryPersistent 将某段内存变为持久内存后，MATLAB 的自动内存管理机制就无法在 MEX 文件结束时释放该段内存了，这时必须通过 mexAtExit 注册一个退出函数，在该退出函数中使用内存释放函数对持久内存进行释放了。

而在其他的 MATLAB 接口应用程序中，函数 mxMalloc 在工作时，将默认地调用 C 语言的库函数 malloc 对内存进行内存分配；如果函数的默认行为产生异常，用户可以定义自己的内存分配函数来替代 malloc，同时通过函数 mxSetAllocFcns 将子定义函数注册为默认的工作函数。

举　例：参见函数 mxMalloc 的范例程序 mxmalloc.c。

75. mxRealloc

功　能：重新分配内存。

语　法：

```
#include "matrix.h"
#include <stdlib.h>
void *mxRealloc(void *ptr, size_t size);
```

说　明：通过函数 mxRealloc，用户可以对某一段内存进行重新分配，void 指针类型的输入参数指明了希望重新分配的内存的起始地址，输入参数 size 则声明了重新分配的内存的大小。如果函数执行失败，函数将返回一个空指针 NULL。但是必须注意的一点是，在这种情况下，用户同样需要对内存进行释放，因为该段内存仍然处于分配状态，如果不进行释放，将导致内存泄漏。

举　例：参见函数 mxSetNzmax 的范例程序 mxsetnzmax.c。

76. mxSetAllocFcns

功　能：在 MAT 文件应用程序和引擎应用程序中，注册自定义的内存分配函数和内存释放函数。

语　法：

```
#include "matrix.h"
#include <stdlib.h>
void mxSetAllocFcns(calloc_proc callocfcn, free_proc freefcn,
                    realloc_proc reallocfcn, malloc_proc mallocfcn);
```

说　明：通过函数 mxSetAllocFcns，用户可以将自定义的内存分配函数和内存释放函数注册到系统中，在 MAT 文件应用程序和引擎应用程序中，供 MAT-

LAB 的接口提供的内存分配函数和释放函数调用。因为在 MAT 文件应用程序和引擎应用程序中，接口函数 mxCalloc、mxFree 和 mxMalloc 仅仅是简单地调用 C 语言的库函数 calloc、free 和 malloc 对内存进行操作，通过编写自定义的内存操作函数，可以令用户完成特定的工作。函数 mxSetAllocFcns 的输入参数的含义分别如下：

- callocfcn 为用户自定义的供函数 mxCalloc 调用的内存分配函数的函数名。通常该函数是以 C 语言的库函数 calloc 为核心，并且其原型必须为如下形式：

  ```
  void * callocfcn(size_t nmemb, size_t size);
  ```

  其中 nmemb 为内存中可以分配的元素的个数，而 size 为每个元素所占用的字节数。此外必须注意的一点是，在用户自定义的内存分配函数 callocfcn 中，必须将所有的内存初始化为 0；
- freefcn 为用户自定义的供函数 mxFree 调用的内存释放函数的函数名。该函数的原型必须为如下形式：

  ```
  void freefcn(void *ptr);
  ```

  其中 ptr 为指向用户希望释放的内存块的指针。此外必须注意的一点是，在用户自定义的内存释放函数 freefcn 中，必须对输入参数 ptr 进行判断，如果 ptr 为 NULL，则立即退出；
- reallocfcn 为用户自定义的供函数 mxRalloc 调用的内存分配函数的函数名。该函数的原型必须为如下形式：

  ```
  void * reallocfcn(void *ptr, size_t size);
  ```

  其中 ptr 为指向用户希望重新分配的内存区域的指针，而 size 为内存区域的大小，单位为字节。
- mallocfcn 为用来替代函数 malloc 进行内存重新分配的函数的函数名。该函数的原型必须为如下形式：

  ```
  void * mallocfcn(size_t n);
  ```

  其中 n 为希望重新分配的内存的大小，单位为字节；此外在函数 mallocfcn 中无需对重新分配的内存进行初始化。

举　例：程序 mxsetallocfcns.c 为 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中，它演示了函数 mxSetAllocFcns 的使用，其源代码如下：

```
/* 头文件包含 */
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"

/* 自定义的内存分配函数 */
```

```
void *my_calloc(size_t n, size_t size)
{
   void   *ptr;

   printf("Calloc Proc. \n");

   if ((n <= 0) || (size <= 0))
     return NULL;
   else
   {
     ptr = calloc(n, size);
     return(ptr);
   }
}

/* 自定义的内存释放函数 */
void my_free(void *ptr)
{
   printf("Free Proc. \n");
   if (ptr == NULL)
     return;
   else
     free(ptr);
}

/* 自定义的供函数 mxMalloc 调用的内存重新分配函数 */
void *my_realloc(void *ptr, size_t size)
{
   printf("Realloc Proc. \n");

   if (size == 0)
     free(ptr);

   if (size <= 0)
     return NULL;

   if (ptr == NULL)
     ptr = malloc(size);
   else
     ptr = realloc(ptr, size);

   return(ptr);
}

/* 自定义的替代 malloc 的内存重新分配函数 */
```

```
void *my_malloc(size_t size)
{
   void    *ptr;

   printf("Malloc Proc. \n");

   if (size <= 0)
     return NULL;
   else
   {
     ptr = malloc(size);
     return(ptr);
   }
}

/* 主函数 */
int main(void)
{
   char *x;
   mxArray *pa;

   /* 注册内存操作函数 */
   mxSetAllocFcns(my_calloc,
     my_free,
     my_realloc,
     my_malloc);

   printf("Creating an array...\n");
   pa = mxCreateString("This is an example of a string.");

   printf("Creating a character buffer...\n");
   x = mxCalloc(255, sizeof(char));

   mxGetString(pa, x, 255);
   printf("String variable contained: %s\n", x);

   printf("Freeing allocated buffer...\n");
   mxFree(x);
   printf("Freeing the array...\n");
   mxDestroyArray(pa);

   return EXIT_SUCCESS;
}
```

77. mxSetCell

功　能:设置单元阵列的单元值。

语　法：#include "matrix.h"

void mxSetCell(mxArray *array_ptr, int index, mxArray *value);

说　明：通过函数 mxSetCell，用户可以对单元阵列的某个单元进行内容设置。函数的三个输入参数的含义分别如下：

- array_ptr 为一个指向单元阵列的指针；
- index 为希望设置的单元的索引值；
- value 为希望设置的值，为一个指向某个阵列的指针。

如果在指定单元已经存在内容，则函数将使用新的内容覆盖原先的内容。

举　例：参见 3.2.4 节的范例程序。

78. mxSetClassName

功　能：将一个结构体阵列转换为对象阵列。

语　法：#include "matrix.h"

int mxSetClassName(mxArray *array_ptr, const char *classname);

说　明：通过函数 mxSetClassName，用户可以将输入参数 array_ptr 指向的结构体阵列转换为由字符串 classname 指定的对象类型的阵列，这主要是为了将阵列内容顺序地存储到 MAT 文件中。如果不通过 MATLAB 的 load 命令将阵列读入到 MATLAB 之前，对象阵列是无效的。如果由 classname 指定的对象类型为 MATLAB 中未定义的对象类型，load 命令将会将对象阵列转换为结构体阵列。如果函数执行成功，将返回 0，否则返回非 0。

79. mxSetData

功　能：设置阵列的数据指针。

语　法：#include "matrix.h"

void mxSetData(mxArray *array_ptr, void *data_ptr);

说　明：通过函数 mxSetData，用户可以设置阵列的数据指针。函数的功能与函数 mxSetPr 的功能极为类似，惟一的不同是函数指向数据的指针为 void 类型。函数的两个输入参数的含义如下：

- array_ptr 为指向某个阵列的指针；
- data_ptr 位数据指针。

80. mxSetDimensions

功　能：更改阵列的维数和各维的大小。

语　法：#include "matrix.h"

int mxSetDimensions(mxArray *array_ptr, const int *dims, int ndims);

说　明：通过函数 mxSetDimensions，用户可以更改阵列的维数和各维的大小，其各输入参数的含义分别如下：

- array_ptr 为一个指向阵列的指针；

- dims 为一个用于指定阵列各维大小的数组，其中 dims[0]代表了阵列的行数，dims[1]代表了阵列的列数，dims[2]代表了阵列的页面数，后续的以此类推；
- ndims 为阵列的维数大小。

如果函数执行成功，函数返回 0，否则返回 1。

81. mxSetField

功　能：设置一个结构体阵列的域的名字和值。

语　法：
```
#include "matrix.h"
void mxSetField(mxArray *array_ptr, int index,
                const char *field_name, mxArray *value);
```

说　明：通过函数 mxSetField，用户可以设置一个结构体阵列的指定域的名字和值，其四个输入参数的含义分别如下：

- array_ptr 为一个指向结构体阵列的指针；
- index 为指定域的索引值；
- field_name 为包含域名的字符串；
- value 为希望设置的值，为一个指向某个阵列的指针。

82. mxSetFieldByNumber

功　能：通过指定元素的索引值和域的索引值，对域的内容进行设置。

语　法：
```
#include "matrix.h"
void mxSetFieldByNumber(mxArray *array_ptr, int index,
                        int field_number, mxArray *value);
```

说　明：通过函数 mxSetFieldByNumber，用户可以通过指定的元素的索引值和域的索引值，对域的内容进行设置。函数的四个输入参数的含义分别如下：

- array_ptr 为一个指向结构体阵列的指针；
- index 为指定阵列的元素的索引值；
- field_number 为指定域的索引值；
- value 为希望设置的值，为一个指向某个阵列的指针。

举　例：参见函数 mxCreateStructArray 的范例程序。

83. mxSetImagData

功　能：设置阵列的虚部数据指针。

语　法：
```
#include "matrix.h"
void mxSetImagData(mxArray *array_ptr, void *pi);
```

说　明：通过函数 mxSetImagData，用户可以对阵列的虚部数据进行设置。输入参数 array_ptr 为一个指向某个阵列的指针，pi 为数据指针。函数的功能与函数 mxSetPi 的功能极为相似，惟一的不同是函数 mxSetImagData 的数据指针为 void 类型。函数 mxSetImagData 可以用于除双精度类型以外的所有数值

阵列的虚数部分赋值。

举　例：参见函数 mxIsFinite 的范例程序。

84. mxSetIr

功　能：设置稀疏矩阵的 ir 数组。

语　法：#include "matrix.h"

```
void mxSetIr(mxArray *array_ptr, int *ir);
```

说　明：通过函数 mxSetIr，用户可以对稀疏矩阵的 ir 数组进行设置。输入参数 array_ptr 为一个指向某个稀疏矩阵的指针，ir 为数据指针。数组 ir 为一个整型数组，其长度为 nzmax，即稀疏矩阵可以包含的最大的非零元素的个数，其元素包含了稀疏矩阵中每个非零元素的行的索引值，并且非零元素在数组按列优先的顺序存放。

举　例：参见 3.2.7 节内容。

85. mxSetJc

功　能：设置稀疏矩阵的 jc 数组。

语　法：#include "matrix.h"

```
void mxSetJc(mxArray *array_ptr, int *jc);
```

说　明：通过函数 mxSetJc，用户可以对稀疏矩阵的 jc 数组进行设置。输入参数 array_ptr 为一个指向某个稀疏矩阵的指针，jc 为数据指针。数组 jc 为一个整型数组，其长度为 n+1，其中 n 为稀疏矩阵的列数，数组内元素的含义可以描述如下：

- jc[j]为第 j 列中包含的第一个非零元素在数组 ir 中的索引值；
- jc[j+1]-1 为第 j 列中包含的最后一个非零元素在数组 ir 中的索引值；
- jc[n+1]为稀疏矩阵中非零元素的个数。

举　例：参见 3.2.7 节内容。

86. mxSetLogical

功　能：设置阵列的逻辑标志。

语　法：#include "matrix.h"

```
void mxSetLogical(mxArray *array_ptr);
```

说　明：通过函数 mxSetLogical，用户可以设置阵列的逻辑标志。当对某个阵列设置逻辑标志后，阵列中所有的非零元素将被认为是逻辑真，而零元素将被认为是逻辑假。

举　例：参见函数 mxIsLogical 的范例程序。

87. mxSetM

功　能：设置阵列的行数。

语　法：#include "matrix.h"

void mxSetM(mxArray *array _ ptr, int m);

说　明：通过函数 mxSetM，用户可以设置由输入参数 array _ ptr 指定的阵列的行数，行数由输入参数 m 确定。这里必须注意的一点是，当使用函数对阵列的行数进行改变后，函数并不对阵列的数据部分包括 pr 和 pi 进行内存的重新分配，所以用户在使用完函数 mxSetM 后，必须使用函数 mxMalloc 对数据内存进行重新分配。

举　例：参见 3.2.7 节范例程序。

88. mxSetN

功　能：设置阵列的列数。

语　法：#include "matrix.h"

void mxSetN(mxArray *array _ ptr, int n);

说　明：通过函数 mxSetN，用户可以设置由输入参数 array _ ptr 指定的阵列的列数，行数由输入参数 n 确定。这里必须注意的一点是，当使用函数对阵列的列数进行改变后，函数并不对阵列的数据部分包括 pr 和 pi 进行内存的重新分配，所以用户在使用完函数 mxSetN 后，必须使用函数 mxMalloc 对数据内存进行重新分配。

举　例：参见 3.2.7 节范例程序。

89. mxSetName

功　能：设置阵列的名字。

语　法：#include "matrix.h"

void mxSetName(mxArray *array _ ptr, const char *name);

说　明：通过函数 mxSetName，用户可以设置由输入参数 array _ ptr 指定阵列的名字，名字由输入参数 name 确定。

举　例：参见 3.2.7 节范例程序。

90. mxSetNzmax

功　能：设置稀疏矩阵所能存放的阵列的非零元素的最大个数。

语　法：#include "matrix.h"

void mxSetNzmax(mxArray *array _ ptr, int nzmax);

说　明：通过函数 mxSetNzmax，用户可以设置由输入参数 array _ ptr 指定的稀疏矩阵所能存放的阵列的非零元素的最大个数，数量值由参数 nzmax 确定。

举　例：程序 mxsetallocfcns.c 为 MATLAB 提供的一个范例程序，存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\MX\

中，它演示了函数 mxSetNzmax 的使用，其源代码如下：

```
/* 头文件包含 */
#include "mex.h"

/* 入口点函数 */
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    int actual_number_of_non_zeros;

    /* 检查输入和输出变量的个数 */
    if (nrhs != 1)
    {
        mexErrMsgTxt("One input argument required.");
    }

    if(nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* 检查输入参数的类型 */
    if(! mxIsSparse(prhs[0]))
    {
        mexErrMsgTxt("Input argument must be sparse\n");
    }

    plhs[0] = mxDuplicateArray(prhs[0]);
    actual_number_of_non_zeros = mxGetJc(plhs[0])[mxGetN(plhs[0])];

    if(mxGetNzmax(plhs[0]) == actual_number_of_non_zeros)
    {
        mexWarnMsgTxt("The actual number of non-zeros is already
            equal to the non-zero maximum for this sparse matrix. \n");
    }
    else
    {
        double *ptr;
        void *newptr;
        int *ir;
        int nbytes;

        nbytes = actual_number_of_non_zeros * sizeof(*ptr);
        ptr = mxGetPr(plhs[0]);
        newptr = mxRealloc(ptr, nbytes);
        mxSetPr(plhs[0], newptr);
```

```
                ptr = mxGetPi(plhs[0]);

                if(ptr != NULL)
                {
                    newptr = mxRealloc(ptr, nbytes);
                    mxSetPi(plhs[0], newptr);
                }

                nbytes = actual_number_of_non_zeros * sizeof(*ir);
                ir = mxGetIr(plhs[0]);
                newptr = mxRealloc(ir, nbytes);
                mxSetIr(plhs[0], newptr);
                mxSetNzmax(plhs[0],actual_number_of_non_zeros);
            }
        }
```

91. mxSetPi

功　能:设置阵列的虚部数据。

语　法:#include "matrix.h"

　　void mxSetPi(mxArray *array_ptr, double *pi);

说　明:通过函数 mxSetPr,用户可以设置输入参数 array_ptr 所指向阵列的虚数部分的数据,数据由输入参数 pi 所指向的双精度类型的内存区域决定。

举　例:参见函数 mxSetNzmax 的范例程序。

92. mxSetPr

功　能:设置阵列的实部数据。

语　法:#include "matrix.h"

　　void mxSetPr(mxArray *array_ptr, double *pr);

说　明:通过函数 mxSetPr,用户可以设置输入参数 array_ptr 所指向阵列的实数部分的数据,数据由输入参数 pr 所指向的双精度类型的内存区域决定。

举　例:参见函数 mxSetNzmax 的范例程序。

93. mxGetClassName

功　能:获得某个阵列的类型。

语　法:#include "matrix.h"

　　const char *mxGetClassName(const mxArray *array_ptr);

说　明:通过函数 mxGetClassName,用户可以获得输入参数 array_ptr 所指向的阵列的类型。

举　例:参见函数 mxIsClass 的范例程序。

第4章 FORTRAN 语言 MEX 文件的编写

FORTRAN 语言 MEX 文件，顾名思义就是基于 FORTRAN 语言编写的 MEX 文件，是 MATLAB 应用程序接口的另一个重要组成部分。通过它不但可以将现有的使用 FORTRAN 语言编写的函数轻松地引入 MATLAB 环境使用，避免了重复的程序设计，而且可以使用FORTRAN 语言为MATLAB 定制用于特定目的的函数，以完成在MATLAB 中不易实现的任务，此外还可以使用 FORTRAN 语言提高 MATLAB 环境中数据处理的效率。

与 C 语言的 MEX 文件相比，它们在功能上相差不大，但是各有长处。使用C 语言编写 MEX 文件的特点是灵活随意，但是必须非常注意 C 语言和 MATLAB 语言中数据存储方式的差别，在 MATLAB 中阵列元素为按列存储，而在 C 语言中数组元素为按行存储；但是如果使用FORTRAN 语言编写MEX 文件就没有这个差别，在FORTRAN 语言中，数组的元素的存储方式同样为按列存储，这主要是因为最初的 MATLAB 是采用 FORTRAN 语言编写的，除此之外，FORTRAN 语言的数值计算功能也非常强大，不足之处是 FORTRAN 与语言中指针的操作较为麻烦，有的编译器甚至不支持指针的操作，这样在使用FORTRAN 语言同MATLAB 环境进行数据交换时就显得麻烦一些，必须进行一些额外的操作，而且FORTRAN 语言有着严格的书写规定。总之，C 语言的 MEX 与 FORTRAN 语言 MEX 文件各有长处，读者可以按照自己的习惯和爱好选择合适的语言。

在上一章中，我们对C 语言 MEX 文件进行了全面的讲解，在本章中我们将对 FORTRAN 语言 MEX 文件进行讲解。首先对FORTRAN 语言 MEX 文件的构成及其执行流程进行说明，然后对 FORTRAN 语言 MEX 文件的编写进行讲述，并给出具体的例子，最后介绍相关的 FORTRAN 语言 MEX 文件的库函数。

4.1 FORTRAN 语言 MEX 文件

4.1.1 一个简单的例子

在开始讲述FORTRAN 语言 MEX 文件之前，先请大家看一个非常简单的样例程序 myplus.f，它定义了两个 DOUBLE PRECISION（双精度类型，在一些编译器上也可以使用REAL * 8 来进行定义）类型数量间的加法运算。该程序的结构非常清晰，极具代表性，请读者仔细阅读，相关内容将在随后的章节中进行介绍。

```
C     myplus.f
C     第一部分：
C     入口子例行程序
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
```

```
C-------------------------------------------------------------------
C     参数声明：
C     plhs (*) 和 prhs (*) 分别为输出和输入参数的指针数组，其中 * 表示数组
C     的长度不定，根据输出和输入参数的个数而定；nlhs 和 nrhs 分别代表输出和
C     输入参数的个数
      integer plhs (*), prhs (*)
      integer nlhs, nrhs
C     调用 API 库函数的声明
      integer mxGetPr, mxCreateFull
      integer mxGetM, mxGetN, mxIsNumeric
C     程序内部使用变量声明
      integer x_pr, y_pr, z_pr
      integer m, n, size
      real * 8 x, y, z
C     检查输入和输出参数的个数：输入参数必须为 2 个，输出参数必须为 1 个，
C     否则程序将终止执行。
      if (nrhs .ne. 2) then
          call mexErrMsgTxt ('Two inputs required!')
      elseif (nlhs .ne. 1) then
          call mexErrMsgTxt ('Too many outputs !')
      endif
C     获取第一个输入参数的维数
      m = mxGetM (prhs (1))
      n = mxGetN (prhs (1))
      size = m * n
C     判断第一个输入参数是否为一个数量
      if (mxIsNumeric (prhs (1)) .eq. 0) then
          call mexErrMsgTxt ('Input one must be a number.')
      endif
C     获取第二个输入参数的维数
      m = mxGetM (prhs (2))
      n = mxGetN (prhs (2))
      size = m * n
C     判断第二个输入参数是否为一个数量
      if (mxIsNumeric (prhs (2)) .eq. 0) then
          call mexErrMsgTxt ('Input two must be a number.')
      endif
C     创建输出参数阵列
      plhs (1) = mxCreateFull (m, n, 0)
C     获取输入和输出参数的数据指针
      x_pr = mxGetPr (prhs (1))
      y_pr = mxGetPr (prhs (2))
      z_pr = mxGetPr (plhs (1))
```

```
C     将数据指针转换为双精度类型数据
      call mxCopyPtrToReal8 (x _pr, x, size)
      call mxCopyPtrToReal8 (y _pr, y, size)

C     调用加法计算子例行程序
      call myplus (x, y, z)

C     将计算所得数据送入到输出数据指针
      call mxCopyReal8ToPtr (z, z _pr, size)

C     返回
      return
      end
C-----------------------------------------------------------------
C     第二部分：
C     加法计算子例行程序
      subroutine timestwo (x, y, z)

C     变量声明
      real * 8 x, y, z

C     加法计算
      z = x + y

C     返回
      return
      end
C-----------------------------------------------------------------
```

对该程序编译后，可以在 MATLAB 下键入以下命令运行：

```
? z = myplus (5, 4)
```

回车后可以得到以下计算结果：

```
z =
   9
```

4.1.2 FORTRAN 语言 MEX 文件源程序的构成

仔细阅读过以上例子程序的读者可能已经发现，FORTRAN 语言 MEX 文件的源程序与C语言MEX 文件源程序的内容大致相同，主要由两个截然不同的部分组成，它们分工明确，分别用于完成不同的任务：

第一部分称为入口子例行程序(gateway routine)，它是计算子例行程序同MATLAB 环境之间的接口，用来完成两者之间的通信任务；

第二部分计算子例行程序(computational routine)，它包含了所有实际完成计算功能的源代码，用来完成实际的计算工作。

1. 入口子例行程序

入口子例行程序的名字为mexFunction，拥有四个虚拟参数，分别为prhs、nrhs、plhs和nlhs，其中prhs 为一个用来存放输入参数地址的整数数组，该数组的数组元素按顺序包含了所有的输入参数的地址，nrhs 为整数类型，它标明了输入参数的个数；plhs 为一

个用来存放输出参数地址的整数数组，该数组的数组元素按顺序包含了所有的输出参数的地址，nlhs 则标明了输出参数的个数，为整数类型。

入口子例行程序的具体的使用格式如下：

```
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
C     虚拟参数声明：
C     plhs (*) 和 prhs (*) 分别为输出和输入参数的指针数组，其中 * 表示数组的长度不
C     定，根据输入和输出参数的个数而定；nlhs 和 nrhs 分别代表输出和输入参数的个数
      integer plhs (*), prhs (*)
      integer nlhs, nrhs
C     调用 API 库函数的声明（因函数使用不同而不同）
      integer mxGetPr, mxCreateFull
        ……
C     一些必要的 C 语言代码，用来完成 MATLAB 与
C     计算子例行程序之间的通信任务
      end subroutine
```

在入口子例行程序中，用户主要可以完成两个方面的任务：一方面，是从输入的参数获得计算所需的数据，然后在用户的计算子例行程序中加以使用，例如在样例程序 myplus.f 中，通过语句 x _ pr = mxGetPr (prhs (1)) 和语句 y _ pr = mxGetPr (prhs (1))，分别从输入的 MATLAB 阵列中得到计算所需的数据指针 x _pr 和 y _pr；另一方面是将计算完毕的结果返回给一个用于输出的数据指针，这样 MATLAB 系统就能够认识从用户计算子例行程序返回的结果。

2. 计算子例行程序

计算子例行程序主要用于完成实际的计算任务，是完全的 FORTRAN 语言编程，不涉及到任何的接口内容。一般来说，这个部分单独编写一个子例行程序或程序子函数，与入口子例行程序分开，在入口子例行程序中调用，但是在计算任务极为简单时，也可以去掉这部分内容将计算代码直接嵌入到入口子例行程序，但是建议最好不要这样，这主要是出于程序的可读性和结构化设计方面考虑。

计算子例行程序既可以象程序 myplus.f 那样存放在一个文件中，也可以单独存放在另外一个文件中，这无关紧要，只不过在编译时略有不同。

3. FORTRAN 语言和 C 语言 MEX 文件的区别

由于 FORTRAN 语言和 C 语言语法上存在较大的差异，所以二者在 MEX 文件的编写上也存在较大的差异，这里有两点值得特别注意：首先，在 C 语言中，字符的大小写是敏感的，所有的函数必须严格按说明使用，不能有任何差别，否则将导致系统报错，而在 FORTRAN 语言中，大小写是不敏感的，名字 mexFunction 与名字 MexFunction、MEXFUNCTION 或 mexfunction 代表的是同一个子例行程序，在样例程序中，我们采用了 mexFunction 这种写法，主要是为了和 C 语言中的函数保持一致，让读者易于阅读和记忆，在下面的讲解中，我们将采用同样的书写方法；其次，在 C 语言 MEX 文件中，有头文件包含语句，用于对头文件进行包含，而头文件对程序中使用的 API 库函数进行了

声明，在 FORTRAN 语言中则没有，但是在程序的开始，即程序的变量声明部分，必须对程序中所使用的函数子程序进行声明，否则在程序中间无法使用，而程序中使用的子例行程序无需声明（有关函数子程序和子例行程序间的区别请读者自行参见有关 FORTRAN 语言语法的书籍）。此外还有一点必须注意，即在 FORTRAN 语言 MEX 文件中只支持两种类型的数据，即双精度类型和字符串类型，而 C 语言 MEX 文件则几乎支持所有的数据类型。

有关 FORTRAN 语言语法方面的知识，不在本书的介绍范围之内，请读者参阅相关书籍。

4.1.3　指针的概念

由于 MATLAB 应用程序接口仅仅能够处理一种类型的数据即 mxArray 结构体，而在 FORTRAN 语言中又没有办法创建一种新的数据类型来与之进行匹配，那么 FORTRAN 语言 MEX 文件是如何与 MATLAB 环境进行交互的呢？答案是指针。熟悉 FORTRAN77 的读者可能会产生这样的疑问，FORTRAN 语言中并没有指针的概念，那又是如何实现指针的传递的呢？基本原理如下：首先，MATLAB 将需要传递的 mxArray 结构体的内存地址作为一个整型数值传递给 FORTRAN 程序；然后在 FORTRAN 程序中，用户通过 MATLAB 应用程序接口函数库提供的访问函数（access routines）使用此整数值来访问 mxArray 结构体的内容，这些访问函数会自动将此整数值作为内存地址，读取相应的内容，如程序 myplus.f 中的语句

```
x_pr = mxGetPr (prhs (1))
call mxCopyPtrToReal8 (x_pr, x, size)
```

完成的功能就是从相应的输入指针中获取了相应的数据，其中 prhs (1) 为第一个输入阵列的地址，在 4.1.1 节的运行命令中，即存储数量 5 的 MATLAB 阵列在内存中的地址，函数 mxGetPr 的功能是从地址 prhs (1) 中获取第一个输入阵列的实部数据地址并存放于整型变量 x_pr 之中，子例行程序 mxCopyPtrToReal8 的功能为从地址 x_pr 之中取出数据存放于双精度型变量 x 之中，这时变量 x 代表的才是真正的数据；最后在计算过程完毕之后，用户同样可以使用相应的访问函数将计算结果输出到一定的内存地址之中，如程序 myplus.f 中的以下三条语句

```
plhs (1) = mxCreateFull (m, n, 0)
z_pr = mxGetPr (plhs (1))
call mxCopyReal8ToPtr (z, z_pr, size)
```

完成的就是数据的输出功能，其中第一条语句的功能是使用库函数 mxCreateFull 构造了一个 $m\times n$ 的满实数矩阵，并将其地址存放于整型的数组元素 plhs (1) 之中，第二条语句的功能为使用函数 mxGetPr 获得地址 plhs (1) 中矩阵的实数部分数据的地址指针，存放于整型变量 z_pr 之中，第三条语句功能为使用子例行程序 mxCopyReal8ToPtr 将计算所得结果 z 输出地址 z_pr 之中，这样就可以使 MATLAB 认识 FORTRAN 程序的输出。在整个操作过程之中，用户无需关心存放 mxArray 结构体地址的整型变量的内容，所有的工作全部交由 MATLAB 提供的接口函数来完成，不过必须注意一点，用户不要随意改变这些整型变量的值，否则会引发意想不到的情况，有可能导致系统的崩溃。

在 FORTRAN90 中，情况发生了一些变化，最为重要的是在 FORTRAN90 中，提

供了对数据指针的支持，用户可以在程序中将一个变量声明为指针类型，这时对数据的访问就可以简单一些了。当用户使用库函数 mxGetPr 获得 mxArray 结构体的数据指针之后，就可以通过%val 直接使用数据而无须调用子例行程序 mxCopyPtrToReal8 和子例行程序 mxCopyReal8ToPtr 了，例如在程序 myplus.f 中可以将语句

```
call myplus (x, y, z)
```

改为语句

```
call myplus (%val (x_pr), %val (y_pr), %val (z_pr))
```

并且可以略去以下三条语句

```
call mxCopyPtrToReal8 (x_pr, x, size)
call mxCopyPtrToReal8 (x_pr, x, size)
call mxCopyReal8ToPtr (z, z_pr, size)
```

以及对变量 x，y 和 z 的声明，而程序完成的功能不变。不过在使用%val 之前必须确认自己的 FORTRAN 编译器支持该函数。

在本章的讲述和举例中，将采用第一种方式，而不是使用%val，为的是适合更多的 FORTRAN 语言编译器。

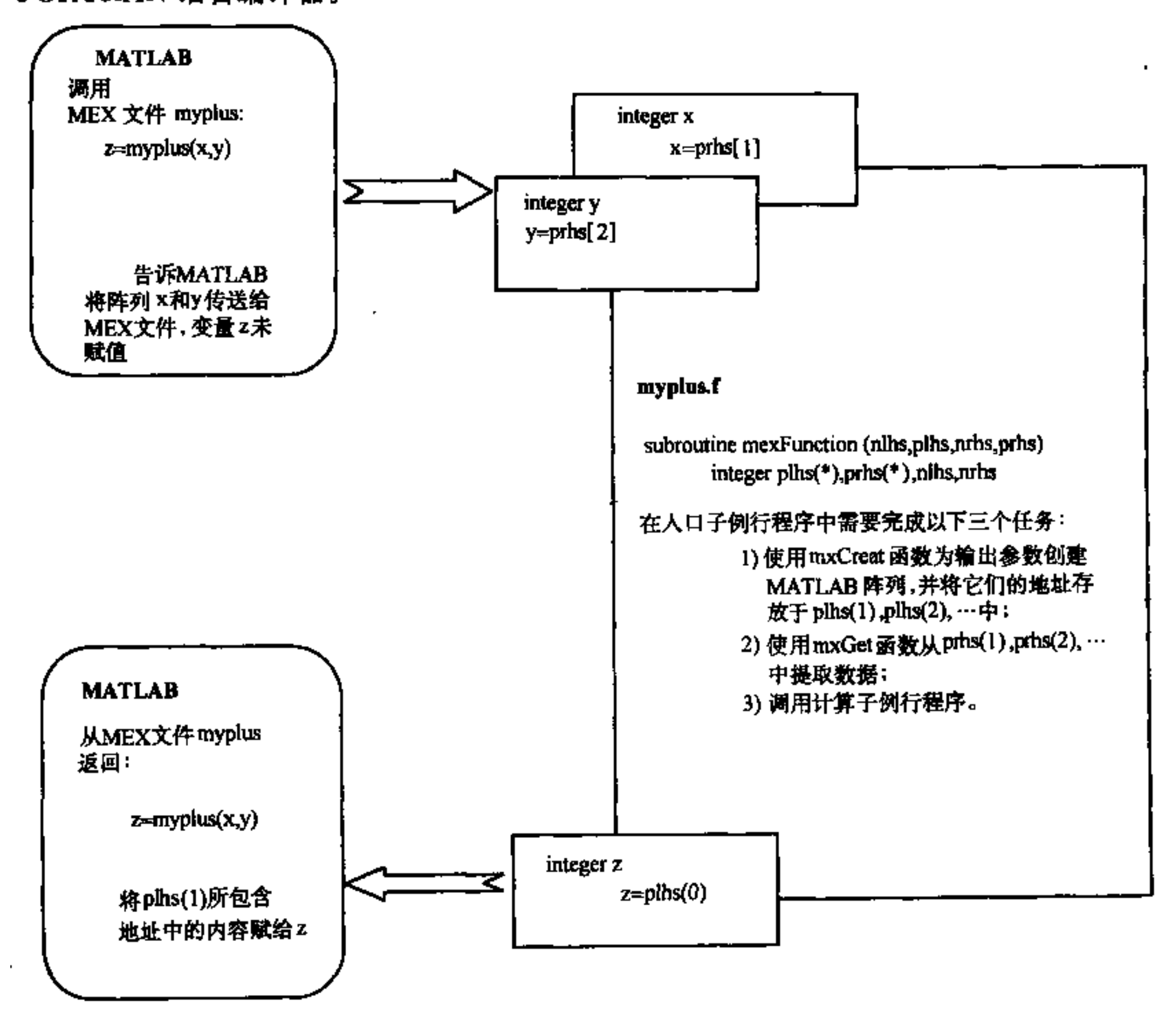


图 4.1 MEX 文件的执行流程图

4.1.4 FORTRAN 语言 MEX 文件的执行流程

当对一个 FORTRAN 语言 MEX 文件的源程序进行编译后，如果成功即可以得到与源程序名相同的 DLL 文件，建议将源程序的取名与程序中计算子例行程序的名字保持相同，这样比较直观而且易于使用和记忆。

在 MATLAB 的工作环境中，按照 MATLAB 语言的语法

```
[a, b, c, …] = mexfile_name (x, y, z, …)
```

正确地键入 MEX 文件名和 MEX 文件所需的参数，就可以运行 MEX 文件了。这时，参数 plhs 和参数 prhs 分别为包含所有输出和输入参数地址的整型数组，参数 nlhs 和 nrhs 则分别包含了输出和输入参数的个数。以程序 myplus.f 为例，当对其进行编译后，可以得到文件名为 myplus.dll 的动态链接程序，在 MATLAB 命令提示符下键入命令

```
z = myplus (x, y)
```

之后，MATLAB 解释器将首先对各参数进行赋值，如下：

nlhs = 1;	nrhs = 2;
plhs (1) 的内容为空；	prhs (1) 包含 MATLAB 阵列 x 的地址；
	prhs (2) 包含 MATLAB 阵列 y 的地址；

然后调用入口子例行程序 mexFunction，来完成计算任务和 MATLAB 与计算子例行程序间的通信。图 4.1 为 MEX 文件的执行流程图。

4.2 FORTRAN 语言 MEX 文件的编程

在第一节中，我们给出了一个用于处理简单数量的 FORTRAN 语言 MEX 程序，本节中我们将基于若干个例子程序，具体讲述如何在 FORTRAN 语言 MEX 文件中对各种类型的 MATLAB 阵列进行处理，使读者对 FORTRAN 语言 MEX 文件的编程有一个全面的了解。

4.2.1 FORTRAN 语言 MEX 文件对字符串的操作

除了双精度类型的数据之外，字符串是 FORTRAN 语言 MEX 支持的另外一种数据类型，在本小节中，我们将对字符串的操作进行讲解。

1. *范例程序*

mystringplus.f

```
C     程序段 (1)
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     程序段 (2)
      integer mxCreateString, mxGetString
      integer mxGetM, mxGetN, mxIsString
C     程序段 (3)
```

```
      integer status, strlen1, strlen2
      character * 100 input_buf1, input_buf2
      character * 200 output_buf

C     程序段 (4)
C     检查输入和输出参数的个数
      if (nrhs .ne. 2) then
         call mexErrMsgTxt ('Two inputs required.')
      elseif (nlhs .ne. 1) then
         call mexErrMsgTxt ('One input required.')
C     输入参数必须为字符串
      elseif (mxIsString (prhs (1)) .ne. 1) then
         call mexErrMsgTxt ('Input must be a string.')
      elseif (mxIsString (prhs (2)) .ne. 1) then
         call mexErrMsgTxt ('Input must be a string.')
C     输入参数必须为行向量
      elseif (mxGetM (prhs (1)) .ne. 1) then
         call mexErrMsgTxt ('Input must be a row vector.')
      elseif (mxGetM (prhs (2)) .ne. 1) then
         call mexErrMsgTxt ('Input must be a row vector.')
      endif

C     程序段 (5)
      strlen1 = mxGetM (prhs (1)) * mxGetN (prhs (1))
      strlen2 = mxGetM (prhs (2)) * mxGetN (prhs (2))
C     程序段 (6)
      status = mxGetString (prhs (1), input_buf1, 100)
      if (status .ne. 0) then
         call mexErrMsgTxt ('String length must be less than 100.')
      endif
C     程序段 (7)
      status = mxGetString (prhs (2), input_buf2, 100)
      if (status .ne. 0) then
         call mexErrMsgTxt ('String length must be less than 100.')
      endif
C     程序段 (8)
      output_buf = ''
C     程序段 (9)
      call mystringplus (input_buf1, strlen1, input_buf2,
     &                   strlen2, output_buf)
C     程序段 (10)
      plhs (1) = mxCreateString (output_buf)
      return
```

```
      end
C     程序段 (11)
      subroutine mystringplus (input_buf1, strlen1, input_buf2,
     &                        strlen2, output_buf)
      integer strlen1, strlen2
      character * (strlen1) input_buf1
      character * (strlen2) input_buf2
      character * (strlen1+strlen2 ) output_buf
C     程序段 (12)
      output_buf = input_buf1 (1: strlen1) // input_buf2 (1: strlen2)
      return
      end
```

2. 程序注释

程序 mystringplus. f 是一个典型的FORTRAN 语言 MEX 源文件，其计算子例行程序和入口子例行程序存放在一个文件之中。它的功能非常简单，用来将两个字符串合二为一。下面按源程序中的编号对程序代码进行解释：

程序段（1）为入口点子例行程序及其形式参数类型的定义，它是 MATLAB 调用 FORTRAN 语言 MEX 文件时的入口点，是所有 FORTRAN 语言 MEX 文件所必须具备的内容；

程序段（2）为一系列的声明语句，对程序中所使用到的 MATLAB 应用程序接口函数库中的函数子程序进行了类型声明；

程序段（3）则对程序中所使用到的一些临时变量进行了声明；

程序段（4）的主要功能是对用户输入和输出参数的个数和类型检查，这是通过函数子程序 mxIsString 以及参数 nlhs 和 nrhs 来完成的，当输入参数的个数不为 2 或者不为字符串类型以及输出参数的个数不为 1 时，程序调用 API 子例行程序 mexErrMsgTxt 输出错误信息，并且终止当前程序的执行；

程序段（5）的作用为通过使用 API 函数子程序 mxGetM 和 mxGetN 获取输入字符串的长度，用于后续的处理；

程序段(6)的功能为通过 API 函数子程序 mxGetString 获取第一个输入字符串阵列的内容，即实际包含的字符串，并且进行判断，用以确定函数子程序 mxGetString 执行得成功与否；

程序段（7）功能同程序段（6），只不过为对第二个输入字符串阵列进行操作；

程序段（8）的作用为初始化输出字符串，将其置为空；

程序段（9）为对计算子例行程序的调用，用于完成两个字符串的连接功能；

程序段（10）调用了 API 的函数子程序 mxCreateString，将计算完成的结果输出到 MATLAB 环境之中；

程序段（11）为计算子例行程序及其形式参数的定义；

程序段（12）为字符串的连接操作。

程序中使用了大量的 API 函数子程序和子例行程序，在本章的最后，我们将对它们

进行详细的说明。

3. 运行结果

对 mystringplus.f 程序进行编译后，可以得到名为 mystringplus.dll 的动态链接库程序，在 MATLAB 命令提示符下键入以下命令：

```
? a = '1234'; b = '5678';
? c = mystringplus (a, b)
```

回车后，可以得到结果

```
c =
    12345678
```

4.2.2 FORTRAN 语言 MEX 文件对矩阵的操作

矩阵是 MATLAB 系统最基本的处理对象，在 FORTRAN 语言的 MEX 文件中，我们不但可以从 MATLAB 环境中接收矩阵对象，而且可以在处理后向 MATLAB 传回矩阵对象。在本小节中，我们将对 FORTRAN 语言 MEX 文件中矩阵的操作进行讲解。

1. 范例程序

```
matrix.f
C     程序段 (1)
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
C--------------------------------------------------------------------
C     形式参数声明
      integer plhs (*), prhs (*)
      integer nlhs, nrhs
C--------------------------------------------------------------------
C     程序段 (2)
      integer mxCreateFull, mxGetPr
      integer mxGetM, mxGetN, mxIsNumeric
C     程序段 (3)
      integer m1, n1, m2, n2, size
      real * 8 x(1000), y(1000), z(1000)
      integer x_pr, y_pr, z_pr
C     程序段 (4)
      if (nrhs .ne. 2) then
          call mexErrMsgTxt ('One input required.')
      elseif (nlhs .ne. 1) then
          call mexErrMsgTxt ('One output required.')
      endif
C     程序段 (5)
      m1 = mxGetM (prhs (1))
      n1 = mxGetN (prhs (1))
      m2 = mxGetM (prhs (2))
```

```
      n2 = mxGetN (prhs (2))
      if (m1 .ne. m2 .or. n1 .ne. n2)    then
         call mexErrMsgTxt ('Row * column must be <= 1000.')
      endif
C     程序段 (6)
      size = m1 * n1
      if (size.gt.1000) then
         call mexErrMsgTxt ('Row * column must be <= 1000.')
      endif
C     程序段 (7)
      if (mxIsNumeric (prhs (1)) .eq. 0) then
         call mexErrMsgTxt ('Input must be a numeric array.')
      endif
      if (mxIsNumeric (prhs (2)) .eq. 0) then
         call mexErrMsgTxt ('Input must be a numeric array.')
      endif
C     程序段 (8)
      plhs (1) = mxCreateFull (m1, n1, 0)
C     程序段 (9)
      x_pr = mxGetPr (prhs (1))
      y_pr = mxGetPr (prhs (2))
      z_pr = mxGetPr (plhs (1))
      call mxCopyPtrToReal8 (x_pr, x, size)
      call mxCopyPtrToReal8 (y_pr, y, size)
C     程序段 (10)
      call matrixplus (x, y, z, m1, n1)
C     程序段 (11)
      call mxCopyReal8ToPtr (z, z_pr, size)

      return
      end
C     程序段 (12)
      subroutine matrixplus (x, y, z, m, n)
      integer m, n
      real * 8 x(m,n), y(m,n), z(m,n)
C     程序段 (13)
      do 20 i=1, m
         do 10 j=1, n
            z(i,j)= x(i,j) + y(i,j)
 10   continue
 20   continue
      return
      end
```

2. 程序注释

程序 matrixplus.f 为一个典型的 FORTRAN 语言 MEX 源文件，其计算子例行程序和入口子例行程序存放在一个文件之中。它的功能非常简单，用于完成两个矩阵之间的相加操作，其中：

程序段 (1) 为入口点子例行程序及其形式参数类型的定义，它是 MATLAB 调用 FORTRAN 语言 MEX 文件时的入口点，是所有 FORTRAN 语言 MEX 文件所必须具备的内容；

程序段 (2) 包含了一系列的声明语句，对程序中所使用到的 MATLAB 应用程序接口函数库中的函数子程序进行了类型声明；

程序段 (3) 对程序中使用到的临时变量进行了声明，其中 x，y 和 z 都声明为大小为 1000 的双精度类型的数组，用于存储输入和输出的矩阵数据；

程序段 (4) 的功能为对输入和输出参数的个数进行检查，程序 matrixplus.f 要求输入的变量数必须为 2，输出的变量数为 1，否则程序将使用 API 子例行程序 mexErrMsgTxt输出一条错误信息，并终止程序的运行；

程序段 (5) 使用 API 函数子程序 mexGetM 和 mexGetN 分别获取了两个输入矩阵的行数和列数，并且进行比较，如果两个输入矩阵的行数和列数不相等，程序将报错，因为矩阵的加法要求两个输入矩阵必须行列相等；

程序段 (6) 对输入矩阵的大小进行计算，并且给出大小限制，如果输入矩阵过大，将报错；

程序段 (7) 用于对输入矩阵的类型进行判断，要求必须为数值类型，这是通过 API 函数子程序 mexIsNumeric 来完成的；

程序段 (8) 调用了 API 函数子程序 mexCreateFull 构造了一个大小为 m1×n1 的实数矩阵，并将该矩阵的内存地址存放于 plhs (1) 之中；

程序段 (9) 的主要功能为获取输入矩阵和输出矩阵的数据指针，并利用这些指针获取实际的数据，这主要是通过函数子程序 mxGetPr 和子例行程序 mxCopyPtrToReal8 来完成；

程序段 (10) 为对计算子例行程序的调用；

程序段 (11) 的功能为将计算的结果输送到输出矩阵中，由子例行程序 mxCopyReal8ToPtr 来完成；

程序段 (12) 为计算子例行程序及其形式参数的定义，变量 x，y 和 z 皆为双精度类型的可调数组；

程序段 (13) 为输入矩阵的加法操作的源代码。

程序中使用到的 API 子例行程序以及函数子程序将在本章的最后进行说明。

3. 运行结果

对 matrixplus.f 程序进行编译后，可以得到名为 matrixplus.dll 的动态链接库程序，在 MATLAB 命令提示符下键入以下命令：

```
? a= [1 2; 3 4];
```

```
? b= [5 6; 7 8];
? c=matrixplus (a, b)
```

回车后，可以得到结果

```
c =
   6     8
   10    12
```

4.2.3　FORTRAN 语言 MEX 文件中对 MATLAB 函数的调用

在 FORTRAN 语言 MEX 文件中，用户不但可以调用 FORTRAN 语言的内建函数，而且还可以调用 MATLAB 的内建函数、运算符、M 文件，甚至可以为其他的 MEX 文件，这是通过 API 提供的子例行程序 mexCallMATLAB 来完成的。在本小节中，我们将对 FORTRAN 语言 MEX 文件中 MATLAB 函数的调用进行讲解。

1. 范例程序

callfunc.f

```
C     程序段 (1)
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     程序段 (2)
      integer lhs (1)
      integer mxIsString, mexCallMATLAB
      integer status
C     程序段 (3)
      if (nrhs .ne. 1) then
          call mexErrMsgTxt ('One Input required.')
      elseif (nlhs .ne. 0) then
          call mexErrMsgTxt ('No Output required.')
      endif
C     程序段 (4)
      if (mxIsString (prhs (1)) .ne. 1) then
          call mexErrMsgTxt ('Input must be a string.')
      endif
C     程序段 (5)
      status = mexCallMATLAB (1, lhs, 1, prhs,'imread')
      if (status .ne. 0) then
          call mexErrMsgTxt ('Something is wrong.')
      endif
C     程序段 (6)
      status = mexCallMATLAB (0, NULL, 1, lhs,'imshow')
      if (status .ne. 0) then
          call mexErrMsgTxt ('Something is wrong.')
```

```
      endif
C     程序段（7）
      call mxFreeMatrix (lhs (1))
      return
      end
```

2. 程序注释

程序 callfunc.f 不是一个典型的 FORTRAN 语言 MEX 文件。它没有包含计算子例行程序部分，所有的操作均在入口点子例行程序中完成。该程序的功能极为简单，它通过 API 函数子程序 mexCallMATLAB 对 MATLAB 图像工具箱中的 M 文件 imread 和 imshow 进行了调用，完成了一幅图像的显示工作，其中源码各个程序段的功能如下：

程序段（1）为入口点子例行程序及其形式参数类型的定义，它是 MATLAB 调用 FORTRAN 语言 MEX 文件时的入口点，是所有 FORTRAN 语言 MEX 文件所必须具备的内容；

程序段（2）为程序中使用到的变量以及 API 函数库中的函数子程序的声明；

程序段（3）为对输入和输出参数个数的检查，程序要求仅有一个输入参数，不需要输出参数；

程序段（4）为输入参数类型的检查，要求输入参数必须为字符串类型；

程序段（5）使用 API 的函数子程序 mexCallMATLAB 对 MATLAB 的 M 文件 imread 进行了调用，读入了一幅图像，同时对函数子程序调用得成功与否进行监控；

程序段（6）使用 API 的函数子程序 mexCallMATLAB 对 MATLAB 的 M 文件 imshow 进行了调用，用于显示程序段（6）中读入的图像，同时对函数子程序调用得成功与否进行监控；

程序段（7）释放了在程序中动态分配的阵列，以防内存泄漏。

3. 运行结果

对 callfunc.f 程序进行编译后，可以得到名为 callfunc.dll 的动态链接库程序，在 MATLAB 命令提示符下键入以下命令：

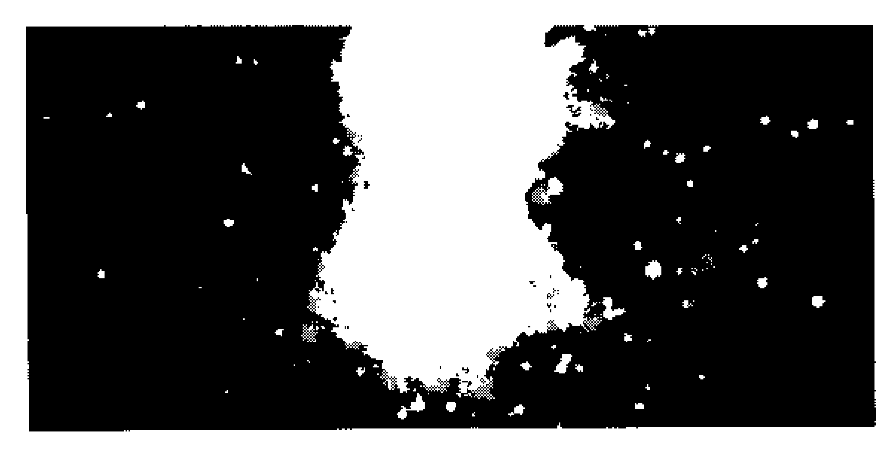


图 4.2　FORTRAN 语言 MEX 文件 callfunc 的执行结果

? callfunc ('ngc4024l. tif')

回车后 MATLAB 将显示图 4.2 中的图形。

4.2.4 FORTRAN 语言 MEX 文件对字符串数组的操作

在 FORTRAN 语言 MEX 文件中，对字符串数组的操作与前面所讲述的对字符串的操作相比存在着较大的差异。

在 MATLAB 环境中，任何数据均以阵列的形式存在，即使是一个数量，在 MATLAB 中也表示为一个 1×1 的阵列，而且在 MATLAB 环境中并没有类似 FORTRAN 语言中字符串数据类型的概念，与之相对应的是字符阵列，在对单独的字符串进行处理时二者差别不大，但是当对象为字符串数组时，二者在存储方式上就存在重大差异了。例如在 FORTRAN 语言中我们定义一个字符串数组

```
character * 5 fstrings (5)
```

并且赋值如下：

```
fstrings (1) = "abcde"
fstrings (2) = "fghij"
fstrings (3) = "klmno"
fstrings (4) = "pqrst"
fstrings (5) = "uvwxy"
```

它们在内存中的存放顺序为

```
abcde fghij klmno pqrst uvwxy
```

接下来我们在 MATLAB 环境中定义一个与字符串数组 fstrings 等价的字符阵列如下：

```
mstrings = ['abcde'; 'fghij'; 'klmno'; 'pqrst'; 'uvwxy'];
```

而它在内存中的存放顺序却为

```
afkpu bglqv chmrw dinsx ejoty
```

显然两者大不相同，这主要是因为：

第一，在 MATLAB 中不存在字符串的概念，字符阵列 mstrings 在 MATLAB 环境中被表示为一个 5×5 的字符阵列，它的每一个元素是一个字符，例如

mstrings (1, 2) = ' b'

第二，MATLAB 中的所有阵列数据的存储方式为按列存储；

第三，FORTRAN 语言中，数组元素的存储方式虽然也为按列存储，但是由于 FORTRAN 语言支持字符串类型的数据，所以在数组 fstrings 元素中包含的字符被视为一个整体，作为一个元素来存放。

因此在将字符串数组进行输出到 MATLAB 环境中或者从 MATLAB 环境中获取字符阵列以填充 FORTRAN 字符串数组时必须进行一些额外的操作。下面我们以一个具体的例子来对 FORTRAN 语言 MEX 文件中字符串数组的操作进行讲解。

1. 范例程序

strings. f

```
C     程序段 (1)
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
```

```
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     第一部分：从 MATLAB 中读取字符矩阵，并存入 FORTRAN 字符串数组
C-------------------------------------------------------------
C     程序段 (2)
      integer mxGetString, mxCreateString
      integer i, j, m, n, size, status, p_str
      character * 150 thestring
      character * 15 string (5), str
C     程序段 (3)
      if (nrhs .ne. 1) then
          call mexErrMsgTxt ('One input required.')
      elseif (nlhs .ne. 1) then
          call mexErrMsgTxt ('One output required.')
      endif
C     程序段 (4)
      m = mxGetM (prhs (1))
      n = mxGetN (prhs (1))
      if (mxIsString (prhs (1)) .ne. 1) then
          call mexErrMsgTxt ('Input must be a char array.')
      elseif (m .gt. 10) then
          call mexErrMsgTxt ('The number of ROWs must be less than 10.')
      elseif (n .gt. 15) then
          call mexErrMsgTxt ('The number of COLs must be less than 10.')
      endif
C     程序段 (5)
      size = m * n
      status = mxGetString (prhs (1), thestring, size)
      if (status .ne. 0) then
          call mexErrMsgTxt ('buffer is too short.')
      endif
C     程序段 (6)
      str=''
      do 10 i =1, m
          do 20 j=1, n
              str (j: j) = thestring (i+ (j-1) *n: i+ (j-1) *n)
   20 continue
      string (i) = str
      call mexPrintf (string (i))
   10 continue
C-------------------------------------------------------------
C
C     第二部分：向 MATLAB 传送字符串
C-------------------------------------------------------------
```

```
C     程序段 (7)
      string (1) = 'MATLAB        '
      string (2) = 'The Scientific        '
      string (3) = 'Computing       '
      string (4) = 'Environment        '
      string (5) = '        by TMW, Inc.'
C     程序段 (8)
      thestring = string (1)
      do 30 i=2, 6
         thestring = thestring (: ( (i-1) * 15)) // string (i)
30    continue
C     程序段 (9)
      p_str = mxcreatestring (thestring)
      call mxSetM (p_str, 15)
      call mxSetN (p_str, 5)
C     程序段 (10)
      call mexCallMATLAB (1, plhs, 1, p_str, 'transpose')
C-----------------------------------------------------------------
      return
      end
```

2. 程序解释

这个程序和我们前面介绍的典型的 FORTRAN 语言 MEX 文件略有不同，它没有计算子例行程序部分，全部的功能均在入口点子例行程序内完成。该程序从总体上可以分为两个部分：第一部分用来从 MATLAB 环境中获取字符阵列，并将该阵列的每一行作为一个字符串赋给一个 FORTRAN 语言的字符串数组；第二部分将一个 FORTRAN 语言的字符串数组输出到 MATLAB 环境成为一个字符阵列。源程序中各程序段的作用如下：

程序段 (1) 为入口点子例行程序及其形式参数类型的定义，它是 MATLAB 调用 FORTRAN 语言 MEX 文件时的入口点，是所有 FORTRAN 语言 MEX 文件所必须具备的内容；

程序段 (2) 对程序中使用的变量和 API 的函数子程序进行了声明，FORTRAN 语言规定所有的说明语句必须出现在执行语句之前；

程序段 (3) 对程序的输入参数和输出参数的个数进行了检查，程序要求输入和输出参数的个数必须为 1，否则将报错退出；

程序段 (4) 对输入参数的维数和类型进行检查，要求输入必须为字符阵列，而且大小必须小于 10×15；

程序段 (5) 通过 API 的函数子程序 mxGetString 获得了输入字符阵列的内容，存放于字符串变量 thestring 之中，同时还对函数子程序 mxGetString 是否成功执行进行判断，并且做出相应的反应；

程序段 (6) 的主要功能为对从 MATLAB 获得的字符串进行重新排列，恢复为行存储的顺序，将结果赋给 FORTRAN 的字符串数组 strings，并使用 API 子例行程序 mex-

Printf 将数组内容输出到 MATLAB 窗口；

程序段（7）对字符串数组进行了重新赋值，用于向 MATLAB 环境输出，从本程序断开始为程序的第二部分；

程序段 (8) 将字符串数组 string 的全部内容合并存放在字符串变量 thestring 中，用于后续的输出；

程序段（9）创建了输出的字符阵列，并且设定了阵列的行数和列数；

程序段 (10) 使用了 API 子例行程序 mexCallMATLAB 调用了 MATLAB 的矩阵转置函数 transpose，用于将字符阵列转置，以符合按列存储的要求，而又不改变字符串的内容。

3. 执行结果

对 strings. f 程序进行编译后，可以得到名为 strings. dll 的动态链接库程序，在 MATLAB 命令提示符下键入以下命令：

```
? mstrings = ['abcde'; 'fghij'; 'klmno'; 'pqrst'; 'uvwxy'];
? a = strings (mstrings)
```

回车后，可以得到结果

```
abcdefghijklmnopqrstuvwxy

a =
MATLAB
The Scientific
Computing
Environment
    by TMW, Inc.
```

4.2.5 包含多个输出的 FORTRAN 语言 MEX 文件

对于包含多个输出的 FORTRAN 语言 MEX 文件的编写，基本上同前面单输出的 FORTRAN 语言 MEX 文件的编写方法相同，惟一需要注意的是参数间的对应关系，例如在 MATLAB 命令提示符下键入以下命令

```
? [a, b, …] = fmex_name (c, d, …)
```

调用 FORTRAN 语言 MEX 文件，这时输出参数 a 的地址包含在 plhs (1) 中，输出参数 b 的地址包含在 plhs (2) 中，后续内容以此类推。下面我们给出一个具体的例子程序 dblout. f，其功能为将两个输入数量的值进行交换，其源代码如下。由于在前面我们已经对 FORTRAN 语言 MEX 文件的编写进行了足够的说明，因此在下面，我们不再对程序进行详细的讲解，而只在程序中加以注释。

```
C     入口点子例行程序的定义
      subroutine mexfunction (nlhs, plhs, nrhs, prhs)
      integer plhs (*), prhs (*)
      integer nlhs, nrhs
C     程序中所使用的 API 函数子程序的声明
      integer mxGetPr, mxCreateFull
      integer mxGetM, mxGetN, mxIsNumeric
```

```
C     程序中所使用变量的声明
      integer pr_in1, pr_in2, pr_out1, pr_out2
      integer m, n, size
      real * 8 x, y
C     对输入和输出变量的个数进行检查
      if (nrhs .ne. 2) then
         call mexErrMsgTxt ('Two inputs required.')
      endif
      if (nlhs .ne. 2) then
         call mexErrMsgTxt ('Two outputs required.')
      endif
C     对第一个输入参数的类型和维数进行检查
      m = mxGetM (prhs (1))
      n = mxGetN (prhs (1))
      if (mxIsNumeric (prhs (1)) .ne. 1) then
         call mexErrMsgTxt ('Inputs must be numeric.')
      elseif (m .ne. 1 .or. n .ne. 1) then
         call mexErrMsgTxt ('Inputs must be scalar.')
      endif
C     对第一个输入参数的类型和维数进行检查
      m = mxGetM (prhs (2))
      n = mxGetN (prhs (2))
      if (mxIsNumeric (prhs (2)) .ne. 1) then
         call mexErrMsgTxt ('Inputs must be numeric.')
      elseif (m .ne. 1 .or. n .ne. 1) then
         call mexErrMsgTxt ('Inputs must be scalar.')
      endif
C     构造输出阵列
      plhs (1) = mxCreateFull (1, 1, 0)
      plhs (2) = mxCreateFull (1, 1, 0)
C     获取输出阵列的数据指针
      pr_in1 = mxGetPr (prhs (1))
      pr_in2 = mxGetPr (prhs (2))
      pr_out1 = mxGetPr (plhs (1))
      pr_out2 = mxGetPr (plhs (2))
C     获取实际的输入数据内容
      size = m * n
      call mxCopyPtrToReal8 (pr_in1, x, size)
      call mxCopyPtrToReal8 (pr_in2, y, size)
C     调用计算子例行程序
         call dblout (x, y)
C     将计算结果送到输出阵列中
      call mxCopyReal8ToPtr (x, pr_out1, size)
```

```
      call mxCopyReal8ToPtr (y, pr_out2, size)
      return
      end
C
C     计算子例行程序的定义
      subroutine dblout (x, y)
      real * 8 x, y, z
C     数据交换
      z = x
      x = y
      y = z
      return
      end
```

对该程序进行编译后，在 MATLAB 命令提示符下键入命令

```
? a = 3; b = 4;
? [c, d] = dblout (3, 4)
```

回车后可以得到如下结果

```
c =
     4
d =
     3
```

4.2.6 FORTRAN 语言 MEX 文件对复数阵列的操作

在 FORTRAN 语言 MEX 文件中，对 MATLAB 复数阵列的操作共有两种方法：第一种为类似 C 语言 MEX 文件中处理复数阵列的方法，使用 API 提供的函数子程序 mxGetPr 和 mxGetPi 分别获得阵列的实部和虚部，进行处理后，再分别使用函数子程序 mxPutPr 和 mxPutPi 将阵列的实部和虚部输出；第二种方法则充分利用了 FORTRAN 语言的优势，即在 FORTRAN 语言存在复数类型的数据，在程序中可以直接使用，API 提供了相应的函数子程序和子例行程序，如 mxCopyPtrToComplex16 等直接将 MATLAB 复数阵列的数据复制到 FORTRAN 语言的复数类型的数组之中，然后进行处理。下面我们给出一个范例程序，同时使用两种方法对复数数据进行处理，程序的功能为完成复数阵列的加法运算，其源代码如下：

```
C     入口点子例行程序的定义
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     程序中所使用的 API 函数子程序的声明
      integer mxGetPr, mxGetPi, mxCreateFull
      integer mxGetM, mxGetN, mxIsComplex
C     程序中所使用变量的声明
      integer mx, nx, my, ny, size
      integer xpr, xpi, ypr, ypi, zpr, zpi
```

```
      complex * 16 x (100), y (100), z (100)
      real * 8 xr (100), xi (100), yr (100), yi (100), zr (100), zi (100)
C     检查输入和输出参数的个数
      if (nrhs.ne.2) then
         call mexErrMsgTxt ('Two inputs required.')
      elseif (nlhs .gt. 2) then
         call mexErrMsgTxt ('Too many output arguments.')
      endif
C     获取输入阵列的维数
      mx = mxGetM (prhs (1))
      nx = mxGetN (prhs (1))
      my = mxGetM (prhs (2))
      ny = mxGetN (prhs (2))
C     限定输入参数的最大行数
      if (mx .gt. 11 .or. my .gt. 11 ) then
         call mexErrMsgTxt ('Inputs rows must be less than 11.')
C     限定输入参数的最大列数
      elseif (nx .gt. 11 .or. ny .gt. 11) then
         call mexErrMsgTxt ('Inputs cols must be less than 11.')
C     检查输入参数是否为复数类型
      elseif ( (mxIsComplex (prhs (1)) .ne. 1) .or.
     &         (mxIsComplex (prhs (2)) .ne. 1)) then
         call mexErrMsgTxt ('Inputs must be complex.')
      endif
C     限定两个输入参数的维数必须相等
      if (mx .ne. my .or. nx .ne. ny) then
         call mexErrMsgTxt ('Inputs must be same size.')
      endif
C     第一种处理方法
C     创建输出复数类型的阵列
      plhs (1) =mxCreateFull (mx, nx, 1)
      size = mx * nx
C     获取输入和输出阵列的部和虚部的数据指针
      xpr = mxGetPr (prhs (1))
      xpi = mxGetPi (prhs (1))
      ypr = mxGetPr (prhs (2))
      ypi = mxGetPi (prhs (2))
      zpr = mxGetPr (plhs (1))
      zpi = mxGetPi (plhs (1))
C     获取实际的输入阵列的数据（实部和虚部）
      call mxCopyPtrToReal8 (xpr, xr, size)
      call mxCopyPtrToReal8 (xpi, xi, size)
      call mxCopyPtrToReal8 (ypr, yr, size)
```

```
      call mxCopyPtrToReal8 (ypi, yi, size)
C     调用计算子例行程序 1
      call plus1 (xr, xi, yr, yi, zr, zi, size)
C     输出阵列
      call mxCopyReal8ToPtr (zr, zpr, size)
      call mxCopyReal8ToPtr (zi, zpi, size)
C     第二种处理方法
C     创建输出复数类型的阵列
      plhs (2) = mxCreateFull (mx, nx, 1)
C     获取输入阵列数据，存放入 FORTRAN 复数数组之中
      call mxCopyPtrToComplex16 (mxGetPr (prhs (1)),
      &                           mxGetPi (prhs (1)), x, size)
      call mxCopyPtrToComplex16 (mxGetPr (prhs (2)),
      &                              mxGetPi (prhs (2)), y, size)
C     调用计算子例行程序 2
      call plus2 (x, y, z, size)
C     输出阵列
      call mxCopyComplex16ToPtr (z, mxGetPr (plhs (2)),
      &                              mxGetPi (plhs (2)), size)
      return
      end
C     计算子例行程序 1
      subroutine plus1 (xr, xi, yr, yi, zr, zi, size)
      real * 8 xr (*), xi (*), yr (*), yi (*), zr (*), zi (*)
      integer size
C     加法操作
      do 11 i=1, size
          zr (i) = xr (i) + yr (i)
          zi (i) = xi (i) + yi (i)
11    continue
      return
      end
C     调用计算子例行程序 2
      subroutine plus2 (x, y, z, size)
      complex * 16 x (*), y (*), z (*)
      integer size
C     加法操作
      do 12 i=1, size
          z (i) = x (i) + y (i)
12    continue
      return
```

```
end
```

对该程序进行编译后，在 MATLAB 命令提示符下键入命令

```
? x = [3-1i, 4+2i, 7-3i];
? y = [8-6i, 12+16i, 40-42i];
? [c, d] =fcomplex (x, y)
```

回车后可以得到如下结果

```
c =
   11.0000 - 7.0000i    16.0000 +18.0000i    47.0000 -45.0000i
d =
   11.0000 - 7.0000i    16.0000 +18.0000i    47.0000 -45.0000I
```

结果完全一样，但从程序可以看出，使用第二种方法明显简单许多。

4.2.7 FORTRAN 语言 MEX 文件对稀疏矩阵的操作

稀疏矩阵是 MATLAB 中一种非常特殊的矩阵类型，它的存储方式与普通的矩阵存在着相当大的差异，因此在操作上也截然不同，下面我们给出一个范例程序，演示如何在 FORTRAN 语言 MEX 文件中对稀疏矩阵进行操作，其功能为将一个普通的矩阵转换为稀疏矩阵。有关稀疏矩阵的存储方式，请读者参见第一章的内容。程序的源代码如下：

```
C     矩阵转换函数子程序定义
      function loadsparse (a, b, ir, jc, m, n)
C     形式参数类型声明
      integer m, n
      integer ir (*), jc (*)
      real*8 a (*), b (*)
C     变量声明
      integer i, j, k
C     矩阵转换，用于填充存储稀疏矩阵的三个阵列的内容
C     这里必须非常注意一点，即在对稀疏矩阵所包含的三个阵列进行索引时，索
C     引的基值为 0，而不是 1。
      k = 1
      do 100 j=1, n
C        初次执行时将 jc 的索引基值设置为 0
         jc (j) = k-1
         do 200 i=1, m
            if (a ( (j-1) *m+i) .ne. 0.0) then
               b (k) = a ( (j-1) *m+i)
C              初次执行时将 ir 的索引基值设置为 0
               ir (k) = i-1
               k = k+1
            endif
200      continue
100   continue
      jc (n+1) = k-1
```

```
C     设定函数子程序的输出值为0
      loadsparse = 0
300   return
      end
C     入口点子例行程序的定义
subroutine mexFunction (nlhs, plhs, nrhs, prhs)
C     子例行程序的形式参数的类型定义
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     对子例行程序中所使用到的 API 函数子程序的说明
      integer mxGetPr, mxCreateSparse, mxGetIr, mxGetJc
      integer mxGetM, mxGetN, mxIsComplex, mxIsDouble
C     变量声明
      integer pr, sr, irs, jcs, lhs (1)
      integer m, n, nzmax, size, nnz
      integer loadsparse
C     定义了四个元素个数可变的数组，用于存储输入和输出的数据
C     该项功能为 FORTRAN90 提供
      real * 8 in (:), out (:)
      integer ir (:), jc (:)
      ALLOCATABLE in, out, ir, jc
C     检查输入和输出参数的个数
      if (nrhs .ne. 1) then
         call mexErrMsgTxt ('One input argument required.')
      endif
      if (nlhs .gt. 1) then
         call mexErrMsgTxt ('Too many output arguments.')
      endif
C     检查输入参数的类型
      if (mxIsDouble (prhs (1)) .eq. 0) then
         call mexErrMsgTxt ('Input argument must be of type double.')
      endif
      if (mxIsComplex (prhs (1)) .eq. 1) then
         call mexErrMsgTxt ('Input argument must be real only')
      endif
C     获取输入矩阵的维数
      m = mxGetM (prhs (1))
      n = mxGetN (prhs (1))
C     使用 ALLOCATE 为大小可变的数组 in 声明大小，并将输入矩阵的数据通过
C     API 提供的函数子程序 mxGetPr 和子例行程序 mxCopyPtrToReal8 存放入数组
C     in 中
      size = m * n
```

```
      ALLOCATE (in (m * n))
      pr = mxGetPr (prhs (1))
      call mxCopyPtrToReal8 (pr, in, size)
C     限定构造的稀疏矩阵的非零元素的最大个数
C     设定为总元素数的 20%
      nzmax = dble (m * n) * .20 + .5
C     调用 MATLAB 函数 find 以及 API 函数子程序 mxGetM 获取输入矩阵中
C     非零元素的个数
      call mexCallMATLAB (1, lhs, 1, prhs,'find')
      nnz = mxGetM (lhs (1))
C     判断输入矩阵中非零元素的个数是否超过设定的最大非零元素的个数，若超
C     过，程序将报错并退出
      if (nnz .gt. nzmax) then
          call mexErrMsgTxt ('Input has too many non-zero elements.')
      endif
C     维数组 out、ir 和 jc 设定大小
      ALLOCATE (out (nnz), ir (nnz), jc (n+1))
C     构造输出的稀疏矩阵，获取该矩阵的数据指针，并通过该指针将数据复制到
C     数组 out 中
      plhs (1) = mxCreateSparse (m, n, nzmax, 0)
      sr = mxGetPr (plhs (1))
      call mxCopyPtrToReal8 (sr, out, nnz)
C     获取稀疏矩阵所包含阵列 ir 的内容
      irs = mxGetIr (plhs (1))
      call mxCopyPtrToReal8 (irs, ir, nnz)
C     获取稀疏矩阵所包含阵列 jc 的内容
      jcs = mxGetJc (plhs (1))
      call mxCopyPtrToReal8 (jcs, jc, n)
C     调用转换函数子程序
      if (loadsparse (in, out, ir, jc, m, n) .eq. 0) then
          call mexPrintf ('Conversion succeed.')
      endif
C     输出稀疏矩阵
      call mxCopyReal8ToPtr (out, sr, nnz)
      call mxCopyReal8ToPtr (ir, irs, nnz)
      call mxCopyReal8ToPtr (jc, jcs, n)
C     释放可变数组使用的内存
      DEALLOCATE (in)
      DEALLOCATE (out, ir, jc)

      return
      end
```

对该程序进行编译后，在 MATLAB 命令提示符下键入命令

```
? a=[5.9, 0, 0, 0, 0; 0, 5.9, 0, 0, 0; 6.2, 0, 0, 0, 0; 0, 6.2, 0, 0, 0; 0, 5.9, 0, 0, 0];
? b=fsparse (a)
```

回车后可以得到如下结果

```
Conversion succeed.
b =
    (1, 1)        5.9000
    (3, 1)        6.2000
    (2, 2)        5.9000
    (4, 2)        6.2000
    (5, 2)        5.9000
```

4.3　FORTRAN 语言 MEX 文件的建立

在 4.1 和 4.2 节中，我们详细讲述了基于 FORTRAN 语言的 MEX 文件的编写。在本节中，我们将对 FORTRAN 语言 MEX 文件的建立进行讲述，使读者全面掌握 FORTRAN 语言 MEX 文件的使用。

4.3.1　FORTRAN 语言 MEX 文件的建立

在第二章第一节对 MEX 文件的简单介绍中，我们提供了一种极为简单的 MEX 文件的建立方法，即直接使用 mex 命令，并在其后加上所希望编译的 MEX 文件的源文件名，具体格式如下：

```
mex <MEX 文件名>
```

其使用条件是已经使用命令

```
mex -setup
```

对默认的选项文件进行了正确的配置。对于一般的应用来说，这已经足够了，但是如果用户希望进行一些高级的操作，例如创建一个新的选项文件用于支持 MATLAB 系统没有直接予以提供支持的编译器，或者对编译过程施加更多的控制，这时使用以上格式就无能为力了。

然而，mex 命令是一个功能非常强大的工具，以上命令格式仅仅是其最简单的一种应用方式，它拥有大量的命令控制参数，通过设置这些参数，允许用户使用不同的手段来完成不同的任务，例如通过使用参数-f，用户可以选择指定的选项文件对源程序进行编译，通过使用参数-g 可以让生成的 MEX 文件包含调试信息，以便进行调试，详细的参数列表以及各参数功能见第三章表 3.1，在 MATLAB 命令提示符下也可以通过键入命令

```
mex -h
```

来获取帮助。

4.3.2　基于 Windows 操作系统的 FORTRAN 语言 MEX 文件的建立流程

在上一小节中，我们对 FORTRAN 语言 MEX 文件的建立进行了总体上的描述，在本小节中，我们将对基于 Windows 操作系统的 FORTRAN 语言 MEX 文件的建立进行详细的讲述。

总的说来，基于 Windows 操作系统的 FORTRAN 语言 MEX 文件的建立流程与 C 语言 MEX 文件的建立流程大致相同，同样可以分为三个步骤，即编译（compiling）、预链接（prelinking）和链接（linking），下面我们分别讲述。

1. 编译（compiling）

在 Windows 操作系统中，整个编译过程必须包含以下步骤：

首先，必须对编译 MEX 文件所使用到的一些环境变量进行设置，包括路径变量 PATH，头文件包含变量 INCLUDE 和库文件变量 LIB，假设用户使用的为 Microsoft Fortran PowerStation 系统，则此时环境变量 PATH、INCLUDE 和 LIB 应分别为 Microsoft Fortran PowerStation 的安装路径及其所包含的头文件和库文件所在的路径。当用户在自己的操作系统或编译器中已经对这些环境变量进行了注册，例如在 Windows 9x 操作系统中，用户已经在 autoexec. bat 中写入这些变量，或者在 Windows NT 操作系统中，在控制面板中的系统选项中的环境变量属性页中定义了这些变量，那么这时用户就可以将选项文件中的这部分内容注释掉，而不会发生问题。如果这些变量设置不正确，在编译 MEX 文件时，系统将报错，说某些文件找不到，导致编译过程的失败退出；

其次，必须定义所使用的编译器的名字 COMPILER，在使用 Microsoft Fortran PowerStation 系统时，COMPILER 应设置为 fl32(注：fl32 为 Microsoft Fortran PowerStation 的编译执行文件)；

再次，必须对编译器的编译标志 COMPFLAGS 进行如下设置：

- 建议使用参数-c，以告诉编译器，只对源文件进行编译而不用链接；
- 必须使用参数-D 定义预处理程序宏 MATLAB _ MAX _ FILE；
- 设置编译器优化参数；
- 设置调试信息参数；
- 自由设置其他一些编译器参数。

2. 预链接（prelinking）

预链接过程主要用来动态创建输入函数库，这些函数库包含了 MATLAB 应用程序接口的库函数，以便在 MEX 文件中使用。所有的 MEX 文件仅仅与 MATLAB 发生链接，与之相关的 .DEF 文件放置于目录

<MATLAB 根目录>\EXTERN\INCLUDE

中。对于 MATLAB，MAT 文件和引擎文件不太相同，在后面章节中将分别讲述。

3. 链接（linking）

整个 MEX 文件的建立过程的最后一个步骤为链接过程，整个过程中必须完成以下操作：

首先，必须定义环境变量 LINKER，用于指明所使用的链接器，在使用 Microsoft Fortran PowerStation 系统时，其定义为：

LINKER = link

其次，必须对链接器进行参数设置，即定义环境变量 LINKFLAGS：

- 使用参数 dll，表明用于创建动态链接库文件；
- 将输出点指向函数 mexFunction；
- 链接在预链接过程产生的输入函数库；
- 自由设置其他一些相关链接器参数。

再次，定义链接优化参数 LINKEROPTIMFLAGS；

第四，定义链接调试信息参数 LINKERDEBUGFLAGS；

第五，在需要的情况下，在环境变量 LINK _ FILE 和 LINK _ LIB 中对链接文件 (link-file) 标示符和链接库 (link-library) 进行定义；

第六，在环境变量 NAME _ OUTPUT 中使用参数 output 建立一个输出标示符和名字，如下：

```
NAME _ OUTPUT=/out:"%OUTDIR%%MEX _ NAME%.dll"
```

其中环境变量 MEX _ NAME 包含了命令行中第一个程序的名字，该环境变量在使用时，参数 output 必须被设置，以使 output 正常工作。如果该环境变量没有被设置，编译器默认地使用命令行中的第一个程序名。

4. 对默认选项文件 mexopts.bat 的分析

选项文件 mexopts.bat 是使用命令

```
mex -setup
```

对系统进行配置时生成的默认的选项文件，其源文件如下，请读者先详细阅读该程序，在后面将对该选项文件进行全面的分析，以加深读者对 FORTRAN 语言 MEX 文件编译过程的理解。

```
@echo off
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rem (1) 普通参数设置
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

set MATLAB=%MATLAB%
set DF _ ROOT=d: \msdev
set VCDir=%DF _ ROOT%\VC
set MSDevDir=%DF _ ROOT%\sharedIDE
set DFDir=%DF _ ROOT%\DF
set PATH=%MSDevDir%\bin;%DFDir%\BIN;%VCDir%\BIN;%PATH%
set INCLUDE=%DFDir%\INCLUDE;%INCLUDE%
set LIB=%DFDir%\LIB;%VCDir%\LIB;%LIB%

rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rem (2) 编译器参数设置
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

set COMPILER=fl32
set COMPFLAGS=-c -G5 -nologo -DMATLAB _ MEX _ FILE
set OPTIMFLAGS=-Ox
set DEBUGFLAGS=-Zi
set NAME _ OBJECT=/Fo
```

```
rem ****************************************
rem (3) 库创建命令 (预编译过程)
rem ****************************************
set PRELINK_CMDS=lib /def:%MATLAB%\extern\include\df50mex.def /machine: ix86
                 /NOLOGO /OUT:%LIB_NAME%1.lib
set PRELINK_DLLS=lib /def:%MATLAB%\extern\include\%DLL_NAME%.def
                 /machine: ix86 /OUT:%DLL_NAME%.lib /NOLOGO
rem ****************************************
rem (4) 链接器参数设置
rem ****************************************
set LINKER=link
set LINKFLAGS =/DLL /EXPORT: _MEXFUNCTION@16
          %LIB_NAME%1.lib /implib:%LIB_NAME%l.lib /NOLOGO
set LINKOPTIMFLAGS=
set LINKDEBUGFLAGS=/debug
set LINK_FILE=
set LINK_LIB=
set NAME_OUTPUT="/out:%OUTDIR%%MEX_NAME%.dll"
set RSP_FILE_INDICATOR=@
rem ****************************************
rem (5) 资源编译器参数设置
rem ****************************************
set RC_COMPILER=rc /fo "%OUTDIR%mexversion.res"
set RC_LINKER=
```

由上面的源代码可以看出，选项文件 mexopts.bat 的结构非常清晰，总共可以分为五个部分：

第一部分为普通参数设置，定义了一些关于 MATLAB 以及 FORTRAN 语言编译器的路径信息，其中%MATLAB%和%DF_ROOT%分别代表了 MATLAB 和 Microsoft Fortran PowerStation 所在的安装路径，其余的一些相关定义，都是建立在它们的基础之上；

第二部分为编译器参数设置，在该选项文件中，通过环境变量 COMPILER 设置了编译器为 fl32，并且通过环境变量 COMPFLAGS、OPTIMFLAGS、和 DEBUGFLAGS 对编译器进行了全面的设置，其中一些参数在本小节的第一部分已经进行了介绍，其余的一些含义如下：

-G5　代表针对奔腾处理器进行优化处理

-Zi　代表使能调试信息

-Ox　代表对源代码进行全面优化

-nologo　代表禁止版权信息

第三部分为库创建命令，即预编译过程，用于创建输入函数库；

第四部分为链接参数设置，在这部分内容中，首先通过环境变量 LINKER 设置了链接器的名字，为 link，然后通过环境变量 LINKFLAGS 设置了链接器的一些参数选项，如

设置了链接为动态链接库文件，使出口点为 mexFunction，同时对于环境变量 LINK _ FILE 和 LINK _ LIB 进行设置，在使用 Microsoft Fortran PowerStation 编译器时可以不用设置；设置环境变量 LINKDEBUGFLAGS 为/debug，表示包含调试信息；

第五部分为资源编译器参数设置，这部分内容将在后面介绍。

4.3.3　将 FORTRAN 语言 MEX 文件与动态链接库 DLLs 链接

将一个动态链接库文件链接到 MEX 文件中，必须完成两方面的工作：首先，必须在命令行中列出动态链接库的文件名；其次，必须正确地设置选项文件中的环境变量 PRELINK _ DLLS，其功能主要是用来动态地从用户输入的动态链接库创建一个输入函数库，以便动态链接库可以被链接到 MEX 文件上。该环境变量包含了产生输入函数库的命令和参数选项，同时使用变量 DLL _ NAME 包含了命令行中提供的动态链接库的文件名，在选项文件 mexopts.bat 中定义如下：

```
set PRELINK _ DLLS=lib /def:%MATLAB%\extern\include\%DLL _ NAME%.def
                    /machine: ix86 /OUT:%DLL _ NAME%.lib /NOLOGO
```

4.3.4　语言 MEX 文件的版本信息

mex 命令可以在建立用户的 MEX 文件时嵌入一个资源文件，该资源文件包含了版本和其他一些基本信息。在 Windows 平台上，该资源文件名为 mexversion.rc，存放在目录

<MATLAB 根目录>\EXTERN\INCLUDE

中。为了在建立 MEX 文件时嵌入这些信息，必须对选项文件进行一定的设置，即上文中选项文件 mexopts.bat 中的第五部分，它包括了对环境变量 RC _ COMPILER 和环境变量 RC _ LINKER 的设置工作，提供了资源编译器和链接器命令。当对编译器命令进行了设置之后，编译后的资源文件将使用标准的链接命令链接到 MEX 文件中；如果同时在选项文件中设置了链接命令，那么资源文件将使用这个命令链接到 MEX 文件中。在选项文件 mexopts.bat 中，只设置了 RC _ COMPILER。

4.3.5　链接多个文件

在建立 FORTRAN 语言 MEX 文件时，不但可以将若干个目标文件结合在一起构成 MEX 文件，而且可以同时包含目标库文件，例如在 Windows 操作系统中，在 MATLAB 命令提示符下键入如下命令

```
? mex title.f text.o name.f thesis.o
```

将产生一个名为 title.dll 的动态链接库文件，可在 MATLAB 工作环境中直接执行。这里必须注意：第一，mex 命令可以操作若干种文件格式，包括 .f，.o 等；第二，在链接多个文件时，生成的 MEX 文件的名字为文件列表中的第一个文件的名字；第三，在将文件列表时，必须写出文件的扩展名，并且用空格分隔。

mex 命令提供的这项功能对于软件开发者来说是非常有用的，它允许用户使用像 MAKE 那样的开发工具来管理包含多程序源文件的 MEX 文件项目，仅仅只需要用户编写一个简单的 MAKEFILE 文件，在这个文件中包含了将用户 MEX 文件项目中各源文

件编译为目标文件的规则，然后就可以激活 mex 命令，将各目标文件结合在一起，从而生成可执行的 MEX 文件。通过这种方法，用户在建立 MEX 文件时，不用每次都对源文件进行编译，而只需链接即可。

4.4　FORTRAN 语言 MEX 文件的调试

对于使用过 FORTRAN 语言等高级计算机语言进行过程序设计的读者来说，可能已经深切地体会到，程序的调试对于一个应用程序的建立来说，具有何等重要的意义。因为对于任何一个程序员来说，一次性地将程序设计完成，并且不发生任何错误，几乎是不可能的事情，往往可能发生一些意想不到的问题，并且这些问题发生在程序的运行过程中，非编译过程和链接过程所能够检测到。而程序的调试却能够帮助程序员在程序的运行过程中找到这些隐含的逻辑错误，并加以改进，是程序编制过程中一个非常重要的步骤。正因为如此，MATLAB 也对 MEX 文件提供了全面的调试功能，包括设置断点、单步运行、变量检查等，而且几乎适用于所有的操作系统。下面我们将对 Windows 操作系统下，FORTRAN 语言 MEX 文件的调试进行说明，整个过程与 Windows 下 C 语言 MEX 文件的调试极为相似，这里必须注明一点，我们所使用的编译器为 Microsoft FORTRAN PowerStation。

如果用户需要对一个 MEX 文件进行调试，首先需要做的就是在使用 mex 命令对源程序进行编译时必须使用-g 命令参数，以使生成的 MEX 文件包含调试信息；然后进入 Microsoft FORTRAN PowerStation 集成调试环境，选取所需要调试的 MEX 文件调入集成环境，同时打开 MEX 文件的源程序；接下来需要做的就是对环境进行设置，只需一步即可完成，选取下拉式菜单 Build 中的菜单项 Settings，选择其中的 Debug 属性页，在 Category 下拉框中选取“General”，并在编辑框 Executable for debug session 中输入 MATLAB 的可执行文件的文件名，必须包含全路径名，点击 OK 按钮，全部的设置工作就宣告完成。下面用户就可以对 MEX 文件进行调试了。至于调试器的使用，请读者自行参见相关帮助。

4.5　Microsoft FORTRAN PowerStation 集成环境中 FORTRAN 语言 MEX 文件的建立

如果读者按照前面讲述的内容对一些例子文件进行了编译和调试，将会发现这种开发的方法非常地不方便，尤其是习惯了使用各种集成开发环境，如 Microsoft Fortran PowerStation 等软件的读者，更会觉得极不适应。建立一个 MEX 文件，首先必须在某个编辑器中进行源代码的编写，然后存盘后回到 MATLAB 的工作环境中，对其进行编译，若发现错误，则必须回到原来的文件编辑器，按照 MATLAB 的错误提示，逐行地对源代码进行查错修改，之后再回到 MATLAB 的工作环境中进行编译，如此往复，直至程序没有错误为止，一个 MEX 文件才宣告完成。整个过程需要来回地在文件编辑器和 MATLAB 工作环境之间切换，非常不方便，而且过程中还没有加入调试步骤，否则将更加麻烦。下面的讲述中，我们将以 Microsoft Fortran PowerStation 的集成环境为对象，介绍

一种在该环境中建立和调试 FORTRAN 语言 MEX 文件的方法。

4.5.1　集成环境中 FORTRAN 语言 MEX 文件的建立步骤

在 Microsoft FORTRAN PowerStation 集成环境中建立 FORTRAN 语言 MEX 文件，可以分为如下一些步骤：

第一步，进入 Microsoft FORTRAN PowerStation 集成环境，选择 File 菜单中的 New 菜单项，将弹出一个 New 选项框，如图 4.3 所示，选择其中的“Project Workspace”项，并点击 OK 按钮；

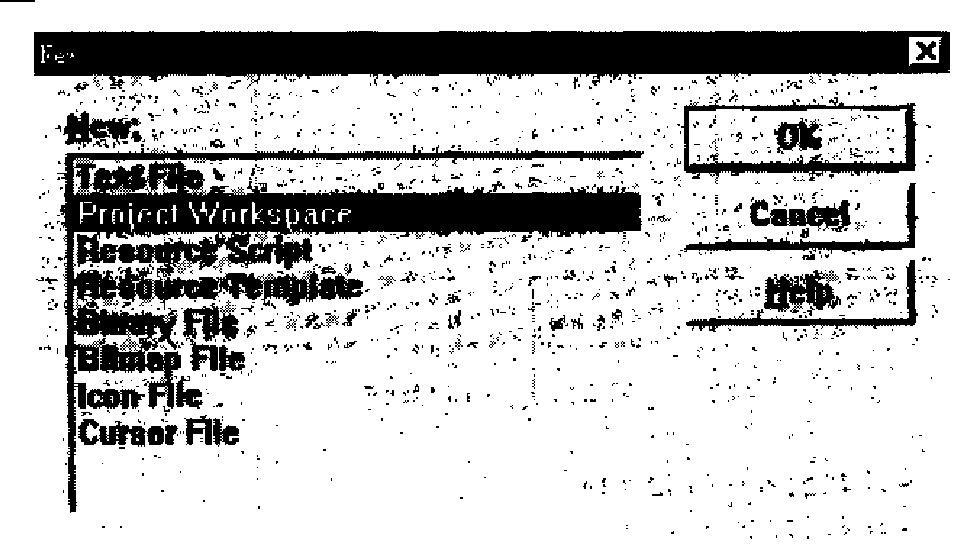


图 4.3　New 选项框

图 4.4　New Project Workspace 选项框

第二步，这时系统将弹出 New Project Workspace 选项框，如图 4.4 所示，在其中的 Type（类型）选项框中将 Project 的类型设置为 Dynamic-Link Library，在 Name 编辑框中输入相应的工程名后，点击 Create 按钮，创建工程；

第三步，在项目工程创建完毕之后，选择下拉式菜单 Tools 中的菜单项 Options，将弹出 Options 对话框，选择其中的 Directories 属性页，如图 4.5 所示，在其中的 Show directories for 下拉式选项框中分别选择 Include Files 和 Library Files，在下部的编辑框中

图 4.5　Options 对话框

输入以下路径：

MATLAB 根目录\EXTERN\INCLUDE

MATLAB 根目录\EXTERN\LIB

然后选择 OK 按钮；

第四步，在 DOS 命令框状态下，进入用户安装 Microsoft FORTRAN PowerStation 的目录，如 d：\msdev，并且进入该目录下的子目录\bin，按下面的格式运行该目录下的命令 lib：

```
lib /DEF:%MATLAB%\extern\include\df50mex.def /MACHINE: IX86 OUT: formex.lib
```

以产生一个名为 formex.lib 的静态链接库文件，其中%MATLAB%表示用户安装 MATLAB 的根目录，如 d：\matlab，一旦库文件 formex.lib 成功生成后，可以应用在所有的 MEX 文件的工程之中，而无须重复生成；

第五步，选择下拉式菜单 Insert 的菜单项 Files into Project，选择库文件 formex.lib 将其嵌入到当前的工程中；同时选择用户的 MEX 文件的源程序，将其也嵌入到当前的工程中；

第六步，在 MEX 源程序的子例行程序 mexFunction 的声明语句后加上如下语句

```
! MS$ATTRIBUTES DLLEXPORT :: MEXFUNCTION
```

用以声明子例行程序 mexFunction 为 MEX 文件的入口点函数，如果不加此语句，源程序也能编译通过，但是在 MATLAB 环境中执行时，MATLAB 将显示如下错误信息

```
??? Invalid MEX-file
```

报告程序为一个非法的 MEX 文件；

第七步，点取下拉式菜单 Build 选择其中的 Settings 菜单项，将弹出一个对话框，选

取其中的 General 属性页（如图 4.6），在 Output files 编辑框中输入一个目录名，编译生成的 MEX 文件将存放在该目录中，建议用户设置为当前工程所在的目录，这样可以为以后的 MEX 文件调试提供方便；

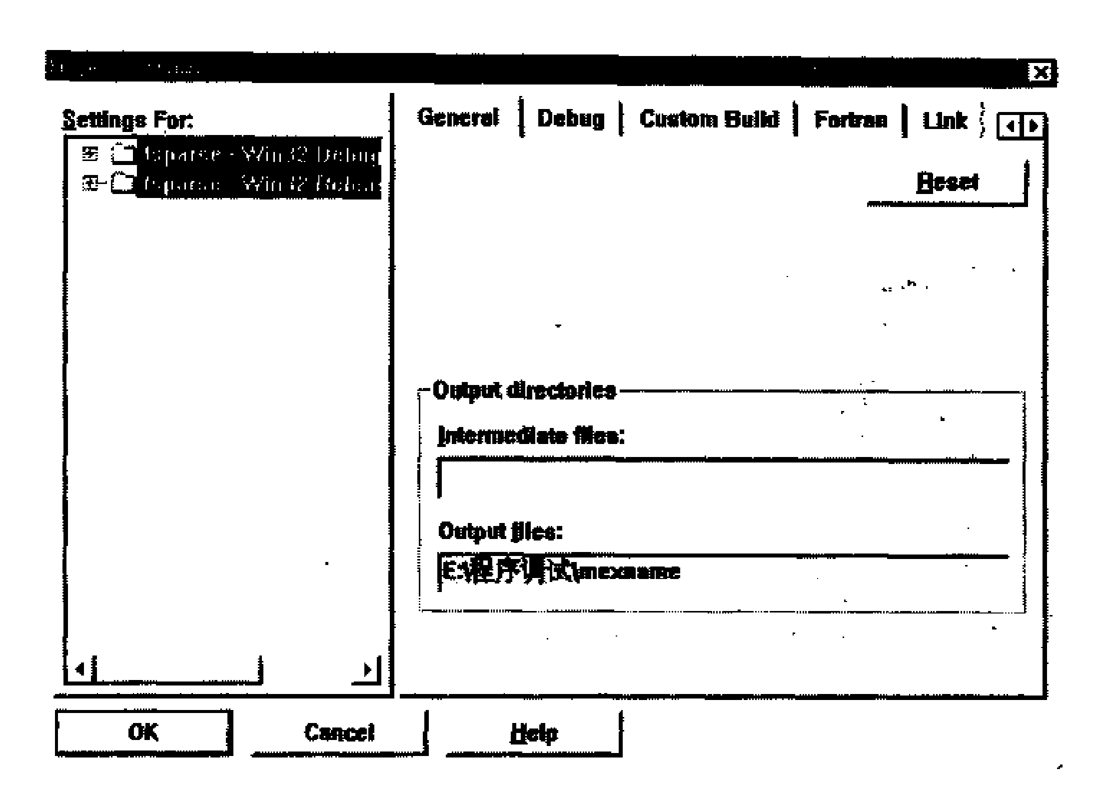


图 4.6　General 属性页

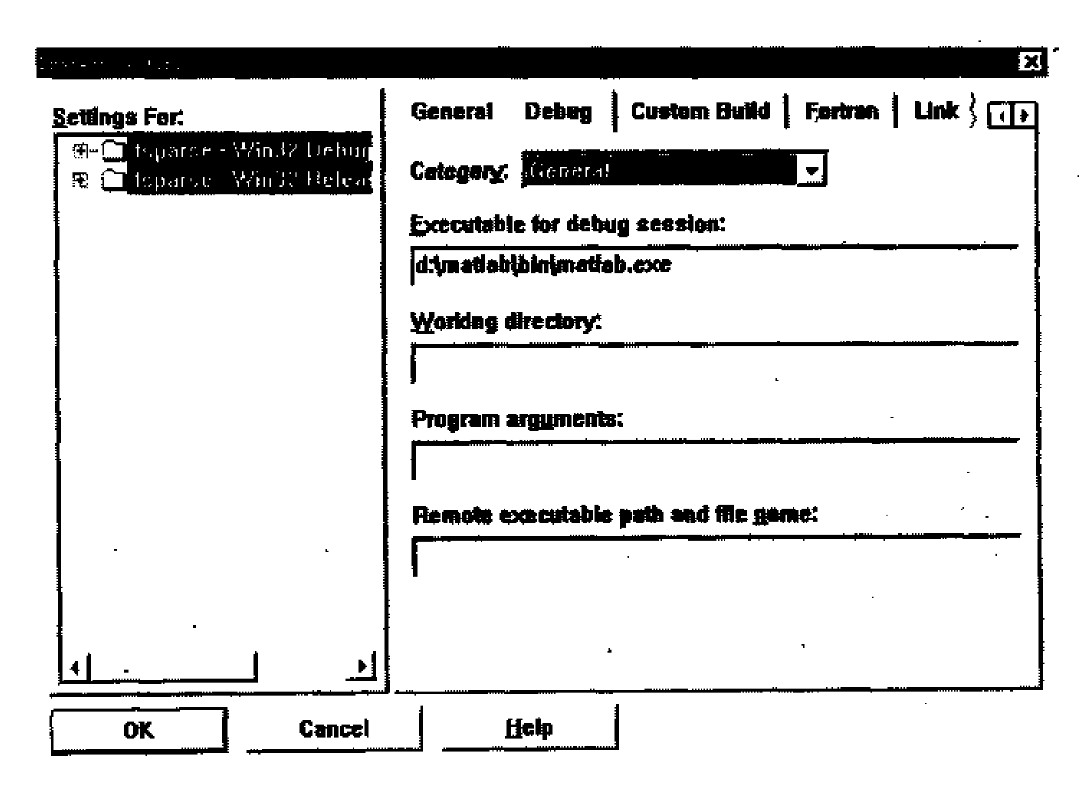


图 4.7　Debug 属性页

第八步，同样在 Settings 菜单项弹出的对话框中，选择其中的 Debug 属性页（如图 4.7），在 Category 下拉框中选取“General”，然后在编辑框 Executable for debug session 中输入 MATLAB 的可执行文件的文件名，必须包含全路径名；

第九步，选择 Project Settings 对话框中的 Fortran 属性页（如图 4.8），在 Category 下拉框中选取“Preprocessor”，然后在 Predefined Preprocessor Variables 编辑框中输入宏名 MATLAB_MEX_FILE；

当完成以上九步工作之后，就可以对 MEX 源程序进行编译和链接，生成可执行的 MEX 文件了。

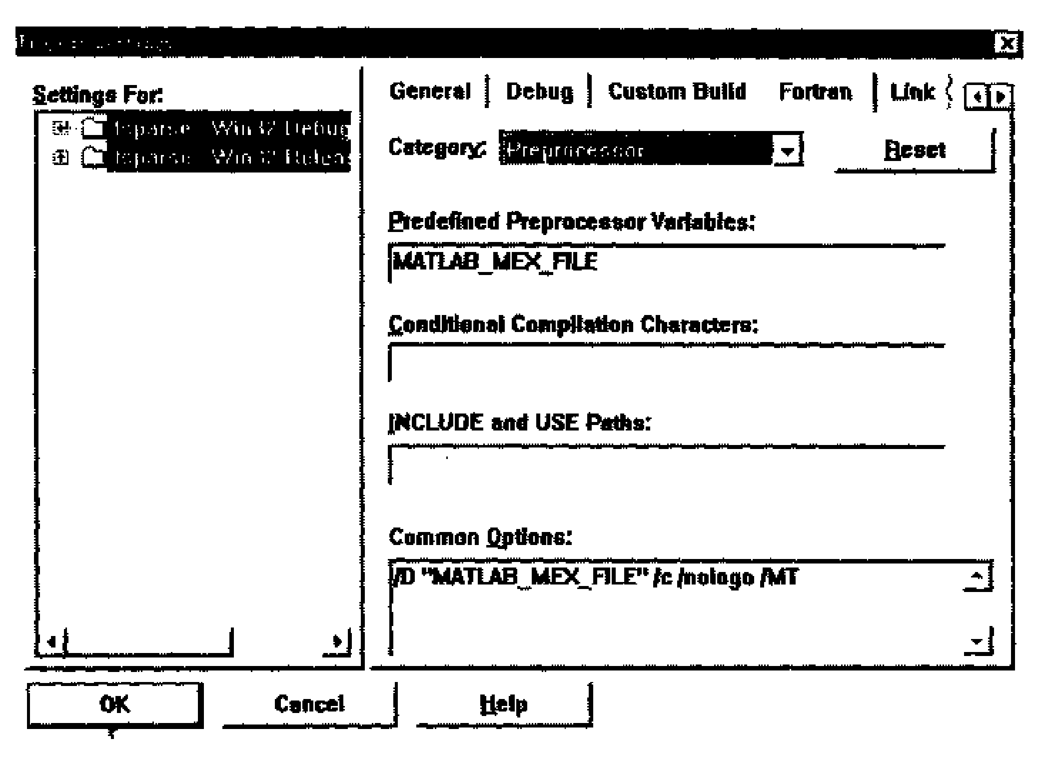


图 4.8　Fortran 属性页

4.5.2　集成环境中 MEX 文件的调试

通过上一小节的设置工作，在 Microsoft Fortran PowerStation 集成环境中对 MEX 文件的调试变得相当简单，基本上没有需要对编译器进行设置的内容，可以像调试一般的应用程序那样进行断点设置、内存读取、单步运行，具体的操作方法请读者参见编译器的相关帮助。

当对 MEX 文件进行调试时，编译器会自动调用 MATLAB，这是在上一小节的第八步中设置，并且 MATLAB 会将当前的工作目录设置为当前工程的路径，这就是在上一小节的第七步中我们建议用户将输出的 MEX 文件路径设置为当前工程目录的原因，这样用户就可以直接在 MATLAB 命令提示符下键入文件名进行运行调试了，否则还需要对 MATLAB 的工作路径进行设置。

4.6　FORTRAN 语言 mex-函数

4.6.1　FORTRAN 语言 mex-函数的声明

在 MATLAB 应用程序接口函数库中，总共提供了 19 个 FORTRAN 语言的 mex-函数，它们的声明分别如下：

integer * 4 function **mexAtExit** (ExitFcn)
integer * 4 function **mexCallMATLAB** (nlhs, plhs, nrhs, prhs, name)
subroutine **mexErrMsgTxt** (error _ msg)
integer * 4 function **mexEvalString** (command)
subroutine **mexFunction** (nlhs, plhs, nrhs, prhs)
real * 8 function **mexGetEps** ()
integer * 4 function **mexGetFull** (name, m, n, pr, pi)
integer * 4 function **mexGetGlobal** (name)
real * 8 function **mexGetInf** ()
integer * 4 function **mexGetMatrix** (name)
integer * 4 function **mexGetMatrixPtr** (name)
real * 8 function **mexGetNaN** ()
integer * 4 function **mexIsFinite** (value)
integer * 4 function **mexIsInf** (value)
integer * 4 function **mexIsNaN** (value)
subroutine **mexPrintf** (message)
integer * 4 function **mexPutFull** (name, m, n, pr, pi)
integer * 4 function **mexPutMatrix** (mp)
subroutine **mexSetTrapFlag** (trap _ flag)

其中，以 function 关键字声明的函数称为函数子程序，而以关键字 subroutine 声明的函数称为子例行程序，在使用时，两者方式略用不同。对于函数子程序来说，它们不但可以通过形实结合来传递返回值，而且可以通过函数名来传递一个返回值，如果在 MEX 文件中希望使用函数子程序函数名的返回值，就必须在 mexFunction 入口点子例行程序中对函数子程序加以声明，而且必须保持名字以及类型完全一致，例如

integer * 4 mexAtExit, mexGetMatrix

对于子例行程序来说，则简单得多，它们仅仅使用形实结合的方式传递返回值，在 MEX 文件中可以直接使用关键字 call 进行调用，而无须任何形式的声明，例如

call mexPrintf ('You are a student.')

4.6.2　FORTRAN 语言 mex-函数的使用说明

下面我们将逐一地讲述 FORTRAN 语言 mex-函数的使用：

1. mexAtExit

功　能：用于登记一个子例行程序，该函数在 MEX 文件被清除或者 MATLAB 终止执行时被调用，用来完成一定的善后工作，如内存释放等。

语　法：integer * 4 function mexAtExit (ExitFcn)

subroutine ExitFcn ()

说　明：函数 mexAtExit 为一个函数子程序，其输入参数 ExitFcn 代表了一个子例行程序名，其返回值永远为零。使用函数 mexAtExit 允许用户登记注册一个自定义 FORTRAN 语言的子例行程序作为退出函数，该函数在 MEX 文件被清除或者 MATLAB 终止执行时被调用，用来完成一定的善后工作，例如释放内存或者关闭文件等等。对于任意一个 MEX 文件来说，只能登记注册一个处于活动状态的退出函数，当一个函数中使用了多次 mexAtExit 函数时，那么只有其中最后使用的一次为有效。此外当一个 MEX 文件处于锁死状态时，所有的试图清除 MEX 文件的操作都将失败，相应的，当一个用户试图清除一个上锁的 MEX 文件时，MATLAB 将不会自动调用退出函数，即使是在一切操作无误的前提下。

举　例：程序 mexatexit.f 为一个使用函数 mexAtExit 的范例程序，其功能极为简单，用于打开一个文件，并写入一定的信息，文件的关闭交由退出函数完成，其源代码如下：

```
C     mexatexit.f
C     入口点子例行程序
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)

C     在 Microsoft Fortran PowerStation 集成环境中编译 MEX 文件时
C     下面的语句用来声明子例行程序 mexFunction 为入口点函数
      MS $ ATTRIBUTES DLLEXPORT :: MEXFUNCTION

C     子例行程序 mexFunction 的形式参数声明
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     声明退出函数 closeup 为外部函数
      EXTERNAL closeup

C     对 MEX 文件中使用的 API 函数进行声明
      integer mxIsString, mxGetString, mxGetM, mxGetN, mexAtExit
      integer mxCalloc

C     程序中使用的变量声明
      integer sta, m, n, strlen
      character * 20 filename
C     声明 num 为一个静态变量
      integer , STATIC:: num

C     检查输入和输出变量的个数
      if (nrhs .ne. 1) then
          call mexErrMsgTxt ('One input argument required.')
      endif
      if (nlhs .ne. 0) then
          call mexErrMsgTxt ('No output argument required.')
      endif
```

```
C     对输入参数的类型进行检查
      sta = mxIsString (prhs (1))
      if (sta .ne. 1) then
         call mexErrMsgTxt ('Input argument must be a string.')
      endif
C     对输出变量的维数进行检查
      m = mxGetM (prhs (1))
      n = mxGetN (prhs (1))
      if (m .ne. 1) then
         call mexErrMsgTxt ('Input argument must be a row vector.')
      elseif (n .gt. 20) then
         call mexErrMsgTxt ('Input argument must be less
     &                             than 20 elements.')
      endif
C     获取输入变量的内容
      strlen = m * n
      sta = mxGetString (prhs (1), filename, strlen)
      if (sta .ne. 0) then
         call mexErrMsgTxt ('String length must be less than 100.')
      endif
C     使用 API 函数子程序 mexAtExit 将子例行程序 closeup 声明为
C     退出函数
      sta = mexAtExit (closeup)
C     将静态变量赋值为 3，用作文件打开编号
      num = 3
C     以 UNKNOWN 方式打开文件 filename
      OPEN (num, FILE= filename, STATUS ='UNKNOWN')
C     在 MATLAB 环境中显示文件打开信息
      call mexPrintf ('File is opened!')
C     向文件写入数据
      WRITE (num, ' (A10)') 'RECORD'
      WRITE (num, ' (I5)') 30303

      return
      end
C     退出函数的声明
      subroutine closeup ()
C     在 MATLAB 环境中显示文件关闭信息
      call mexPrintf ('File is closed!')
C     关闭文件
      CLOSE (num)
      end
```

对该程序编译后在 MATLAB 命令提示符下键入如下命令

```
? mexatexit ('ccc.txt')
```

回车后 MATLAB 将显示

```
File is opened!
```

再键入命令

```
? clear mexatexit
```

回车后 MATLAB 将显示

```
File is closed!
```

退出函数调用成功。

2. mexCallMATLAB

功　能：用于调用 MATALB 的内建函数、用户自定义的 MATLAB M 文件以及 MEX 文件。

语　法：integer * 4 function mexCallMATLAB (nlhs, plhs, nrhs, prhs, name)
integer * 4 nlhs, nrhs, plhs (*), prhs (*)
character * (*) name

说　明：函数 mexCallMATLAB 为一个函数子程序，通过它，用户可以调用 MATLAB 环境中的各种可执行程序，包括 MATLAB 的各种内建函数、运算符，用户自定义的 MATLAB M 文件以及 MEX 文件，它的四个形式参数的含义分别如下：

- name 为用户希望执行的 MATLAB 内建函数或其他一些文件和命令的名字，为字符串类型；
- nlhs 为用户执行 name 所包含的命令时期望输出的参数的个数，为整数类型，系统规定 nlhs 必须小于等于 50；
- plhs (*) 为包含命令 name 输出参数内存地址的整数类型的数组，其中数组元素 plhs (1) 包含了 name 第一个输出参数的内存地址，数组元素 plhs (2) 包含了 name 第二个输出参数的内存地址，下面依次类推，数组元素 plhs (nlhs) 包含了 name 第 nlhs 个输出参数的内存地址；
- nrhs 为用户期望执行的命令 name 的输入元素的个数，为整数类型，系统规定 nrhs 必须小于等于 50；
- prhs (*) 为包含命令 name 输入参数内存地址的整数类型的数组，其中数组元素 prhs (1) 包含了 name 第一个输入参数的内存地址，数组元素 prhs (2) 包含了 name 第二个输入参数的内存地址，下面依次类推，数组元素 prhs (nrhs) 包含了 name 第 nrhs 个输入参数的内存地址；

如果函数子程序 mexCallMATLAB 调用的命令正确执行，返回值为零；若运行发生错误，返回值为非零。在默认情况下，调用命令发生错误时，MATLAB 系统将终止当前 MEX 文件的运行，并返回到 MATLAB 命令提示符下，如果用户希望在发生错误时进行自定义的错误处理，可以使用子例行程序 mexSetTrapFlag，该子例行程序将在后面介绍。

举 例：程序 mexcallmatlab.f 为一个使用函数 mexCallMATLAB 的范例程序，其功能为调用 MATLAB 的内建函数 eig 计算一个输入矩阵的特征向量和特征值，并且调用内建函数 disp 显示结果，程序源代码如下：

```
C      mexcallmatlab.f
C      入口点子例行程序声明及形参类型说明
       subroutine mexFunction (nlhs, plhs, nrhs, prhs)
       ! MS $ ATTRIBUTES DLLEXPORT :: MEXFUNCTION
       integer nlhs, nrhs
       integer plhs (*), prhs (*)
C      对 MEX 文件中使用的 API 函数进行声明
       integer mxIsDouble, mxGetM, mxGetN
       integer mxCreateFull, mxGetPr
C      程序中使用变量的声明
       integer sta, m, n, lhs (2)
       integer ptr1, ptr2, ptr3, size
       real * 8 realdata (1000)
C      检查输入和输出参数的个数
       if (nrhs .ne. 1) then
           call mexErrMsgTxt ('One input argument required.')
       endif
       if (nlhs .ne. 2) then
           call mexErrMsgTxt ('Two output arguments required.')
       endif
C      检查输入参数的类型
       sta = mxIsDouble (prhs (1))
       if (sta .ne. 1) then
           call mexErrMsgTxt ('Input argument must be double.')
       endif
C      检查输入矩阵的大小
       m = mxGetM (prhs (1))
       n = mxGetN (prhs (1))
       size = m * n
       if (size .gt. 1000) then
           call mexErrMsgTxt ('Input matrix is too big.')
       endif
C      创建输出矩阵
       plhs (1) = mxCreateFull (m, n, 0)
       plhs (2) = mxCreateFull (m, n, 0)
C      获取两个输出矩阵的实部数据指针
       ptr1 = mxGetPr (plhs (1))
       ptr2 = mxGetPr (plhs (2))
C      调用 MATLAB 内建函数 eig 和 disp 计算和显示输入矩阵的特征
```

```
C       向量和特征值
        call mexCallMATLAB (2, lhs, 1, prhs, "eig");
        call mexCallMATLAB (0, NULL, 1, lhs (1), "disp");
        call mexCallMATLAB (0, NULL, 1, lhs (2), "disp");

C       输出特征向量矩阵
        ptr3 = mxGetPr (lhs (1))
        call mxCopyPtrToReal8 (ptr3, realdata, size)
        call mxCopyReal8ToPtr (realdata, ptr1, size)

C       输出特征值矩阵
        ptr3 = mxGetPr (lhs (2))
        call mxCopyPtrToReal8 (ptr3, realdata, size)
        call mxCopyReal8ToPtr (realdata, ptr2, size)

C       释放内存
        call mxFreeMatrix (lhs (1))
        call mxFreeMatrix (lhs (2))

        return
        end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? a= [1 2 3; 4 5 6; 7 8 9];
? [b, c] =mexcallmatlab (a)
```

回车后 MATLAB 将显示如下结果

```
   0.2320      0.7858      0.4082
   0.5253      0.0868     -0.8165
   0.8187     -0.6123      0.4082

  16.1168           0           0
        0     -1.1168           0
        0           0     -0.0000

b=
   0.2320      0.7858      0.4082
   0.5253      0.0868     -0.8165
   0.8187     -0.6123      0.4082

c=
  16.1168           0           0
        0     -1.1168           0
        0           0     -0.0000
```

其中第一和第二个矩阵为命令 disp 显示，b 和 c 为计算结果输出，可以通过在命令后使用分号加以屏蔽。

3. mexErrMsgTxt

功　能：用于输出错误信息，并返回到 MATLAB 命令提示符下。

语　法：subroutine mexErrMsgTxt (error _ msg)

character * (*) error _ msg

说　明：函数 mexErrMsg Txt 为一个子例行程序，在 MEX 文件中可以直接使用 call 关键字调用，其输入参数为一个字符串。该函数一般用于当检查到 MEX 文件的某一执行条件没有得到满足时，输出一个错误信息，并且终止当前 MEX 文件的执行，返回到 MATLAB 命令提示符下。

举　例：

```
if (nrhs .ne. 1) then
    call mexErrMsgTxt ('One input argument required.')
endif
if (nlhs .ne. 0) then
    call mexErrMsgTxt ('No output argument required.')
endif
```

该段程序在前面的范例程序中经常出现，用于判断输入和输出参数的个数是否正确，若不正确，则调用子例行程序输出错误信息，并终止 MEX 文件的执行。

4. mexEvalString

功　能：用于输入一个表达式命令到 MATLAB 工作环境中执行。

语　法：

```
integer * 4 function mexEvalString (command)
character * (*) command
```

说　明：函数 mexEvalString 为一个函数子程序，其输入参数为一个字符串，该字符串代表了一个可以在 MATLAB 工作环境中执行的 MATLAB 命令，如果该命令执行成功，其返回值为 0，若不成功，则返回一个非零的整数值。该函数与前面讲述的函数 mexCallMATLAB 的功能大致相同，惟一不同的是函数 mexCallMATLAB 不但可以用来执行 MATLAB 命令，而且可以通过参数 nlhs 和参数 plhs 从 MATLAB 环境中得到计算的结果，用于 MEX 文件中的后续计算；而函数 mexEvalString 则没有此项功能，只能用于向 MATLAB 发送计算指令，无法取得计算结果，并且出现于命令右侧的参数必须在当前的 MEX 文件的工作空间中已经存在。

举　例：程序 mexevalstring.f 为一个使用函数 mexEvalString 的范例程序，它没有任何的输入参数，其功能为使用函数 mexEvalString 关闭 MATLAB 的报警功能，然后对表达式 0/0 进行计算，在没有关闭 MATLAB 报警功能的情况下，MATLAB 将显示一个被零除的警告，而关闭后，MATLAB 将不显示该警告。程序的源代码如下：

```
C     mexevalstring.f
C     入口点子例行程序声明及形参类型说明
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
        ! MS$ATTRIBUTES DLLEXPORT :: MEXFUNCTION
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     检查输入和输出参数个数
      if (nrhs .ne. 0) then
```

```
            call mexErrMsgTxt ('No input argument required.')
        endif
        if (nlhs .ne. 0) then
            call mexErrMsgTxt ('No output argument required.')
        endif
C       使用函数 mexEvalString 传送命令 a=0/0，此时 MATLAB 执行
C       该命令将报警
        call mexEvalString ('a=0/0')
C       关闭报警功能
        call mexEvalString ('warning off')
C       在此执行命令 a=0/0，此时 MATLAB 将不报警
        call mexEvalString ('b=0/0')
C       开启报警功能
        call mexEvalString ('warning on')
        return
        end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? mexevalstring
```

回车后 MATLAB 将显示如下内容

```
        Warning: Divide by zero.
a =
        NaN
b =
        NaN
```

其中 NaN 的含义为 not-a-number。

5. mexFunction

功　能：FORTRAN 语言 MEX 文件的入口点函数

语　法：subroutine mexFunction (nlhs, plhs, nrhs, prhs)

integer * 4 nlhs, nrhs, plhs (*), prhs (*)

说　明：函数 mexFunction 为一个子例行程序，其四个形式参数的含义分别如下：

- nlhs 为 MEX 文件的输出参数个数
- plhs (*) 为包含输出参数内存地址的整数数组
- nrhs 为 MEX 文件的输入参数的个数
- prhs (*) 为包含输入参数内存地址的整数数组

函数 mexFunction 并不是一个由用户来调用的子例行程序，它是所有 FORTRAN 语言 MEX 文件的不可或缺的组成部分，是所有 MEX 文件的入口点函数，通过 MATLAB 系统来调用。当在 MATLAB 环境中执行一个 MEX 文件时，MATLAB 首先将寻找并载入同名的 MEX 文件，然后在该文件中寻找名为 mexFunction 的符号名，如果找到，那么 MATLAB 将以该符号的地址作为调用 MEX 文件的入口地址，并且将函数 mexFunction

的各参数自动赋值；如果在整个文件中没有找到名为 mexFunction 的符号，那么 MATLAB 将否认这是一个 MEX 文件并给出相应的错误信息。

举　例：函数 mexFunction 是 FORTRAN 语言 MEX 文件的编程基础，深入理解该函数对以后 MEX 文件的编程极为重要。程序 mexfunction.f 是一个范例程序，它完成的功能相当简单，仅仅是读取输入参数的内容，并原封不动地赋值给输出参数，其源代码如下：

```
C      mexfunction.f
C      入口点子例行程序声明及形参类型说明
       subroutine mexFunction (nlhs, plhs, nrhs, prhs)
       ! MS$ATTRIBUTES DLLEXPORT :: MEXFUNCTION
       integer nlhs, nrhs
       integer plhs (*), prhs (*)
C      对 MEX 文件中使用的 API 函数进行声明
       integer mxIsDouble, mxIsString, mxIsComplex
       integer mxGetM, mxGetN, mxGetPr, mxGetPi
       integer mxGetString, mxCreateString
C      程序中使用变量的声明
       integer typenum (100), i, j
       integer ptr_inx, ptr_outx, ptr_iny, ptr_outy, m, n, size
       real*8 x (1000), y (1000)
       character*100 string
C      检查输入和输出参数的个数
       if (nrhs .eq. 0) then
           call mexErrMsgTxt ('Input arguments required.')
       endif
       if (nlhs .eq. 0) then
           call mexErrMsgTxt ('Output arguments required.')
       endif
       if (nlhs .ne. nrhs) then
           call mexErrMsgTxt ('Number of output arguments must
     &                       equal number of input arguments.')
       endif
C      检查所有输入参数的类型，并记录
       do 10 i = 1, nrhs
           if (mxIsDouble (prhs (i)) .eq. 1) then
               typenum (i) = 1
           endif
           if (mxIsComplex (prhs (i)) .eq. 1) then
           typenum (i) = 2
           endif
           if (mxIsString (prhs (i)) .eq. 1) then
               typenum (i) = 3
           endif
```

```
10    continue
C     对输出参数进行与输入参数相应的类型设置并赋值
      do 20 j = 1, nrhs
         select case (typenum (j))
C     当输入参数为双精度类型的实数时:
            case (1)
               m = mxGetM (prhs (j))
               n = mxGetN (prhs (j))
               size = m * n
               if (size .gt. 1000) then
                  call mexErrMsgTxt ('Input argument is too big.')
               endif
               plhs (j) = mxCreateFull (m, n, 0)
               ptr_inx = mxGetPr (prhs (j))
               ptr_outx = mxGetPr (plhs (j))
               call mxCopyPtrToReal8 (ptr_inx, x, size)
               call mxCopyReal8ToPtr (x, ptr_outx, size)
C     当输入参数为双精度类型的复数时:
            case (2)
               m = mxGetM (prhs (j))
               n = mxGetN (prhs (j))
               size = m * n
               if (size .gt. 1000) then
                  call mexErrMsgTxt ('Input argument is too big.')
               endif
               plhs (j) = mxCreateFull (m, n, 1)
               ptr_inx = mxGetPr (prhs (j))
               ptr_outx = mxGetPr (plhs (j))
               ptr_iny = mxGetPi (prhs (j))
               ptr_outy = mxGetPi (plhs (j))
               call mxCopyPtrToReal8 (ptr_inx, x, size)
               call mxCopyReal8ToPtr (x, ptr_outx, size)
               call mxCopyPtrToReal8 (ptr_iny, y, size)
               call mxCopyReal8ToPtr (y, ptr_outy, size)
C     当输入参数为字符串类型时:
            case (3)
               m = mxGetM (prhs (j))
               n = mxGetN (prhs (j))
               size = m * n
               if (m .ne. 1) then
                  call mexErrMsgTxt ('String must be a row vector')
               elseif (n .gt. 1000) then
                  call mexErrMsgTxt ('String is too long.')
               endif
```

```
                    if (mxGetString (prhs (j), string, size) .ne. 0) then
                        call mexErrMsgTxt ('error')
                    endif
                    plhs (j) = mxCreateString (string)
                END SELECT
    20      continue
            return
                end
```

对该程序编译后，在MATLAB命令提示符下键入以下命令

```
? a=rand (3, 3);
? b='Welcome!';
? c=125;
? [d, e, f] =mexfunction (a, b, c)
```

回车后MATLAB将显示如下结果

```
d=
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
e =
Welcome!
f =
    125
```

6. mexGetEps

功　能：获得eps的值。

语　法：real * 8 function mexGetEps ()

说　明：函数mexGetEps是一个函数子程序，它没有任何的输入参数，其返回值为MATLAB环境中变量eps的值，该值为1除以MATLAB环境中最大的浮点数的结果，在MATLAB命令提示符下键入以下命令

```
? eps
```

回车后可以得到如下的结果

```
ans =
2.2204e-016
```

eps作为一种精度的衡量标准，在许多MATLAB的内建函数中得到广泛的应用，例如在求矩阵的广义逆函数pinv中，默认的误差限就为eps，当求出的值小于eps时，就认为该值为0；在求矩阵秩的函数rank中，也以eps为默认的判0误差限。

举　例：函数子程序mexGetEps的使用方法极为简单，不过在使用前必须首先在程序的说明语句段进行声明，如下：

```
real * 8 eps
real * 8 mexGetEps
```

下面就可以直接在程序执行语句中使用了，格式如下：

eps = mexGetEps ()

7. mexGetFull

功　能：用于从 MATLAB 工作空间中获得一个双精度类型阵列的全部数据，包括实部和虚部。

语　法：integer * 4 function mexGetFull (name, m, n, pr, pi)

integer * 4 m, n, pr, pi

character * (*) name

说　明：函数 mexGetFull 为一个函数子程序，其五个形式参数的含义分别如下：

- 形式参数 name 为希望获取阵列的名字，是一个字符串变量；
- 形式参数 m 为 name 代表的阵列的行向量数，为整型变量；
- 形式参数 n 为 name 代表的阵列的列向量数，为整型变量；
- 形式参数 pr 为 name 代表阵列的实部数据的指针，为整型变量；
- 形式参数 pi 为 name 代表阵列的虚部数据的指针，为整型变量。

该函数子程序为用户提供了一种直接从 MATLAB 工作空间获得双精度类型阵列的方法，它通过 mxArray 结构体，将阵列的维数和数据指针存放入变量 m、n、pr 和 pi 中，接下来用户通过使用 API 函数 mxCopyPtrToReal8 就可以获得变量 pr 和 pi 所存储内存地址的数据了。在阵列为实数类型时，变量 pi 中虚部数据的指针为 0。如果函数子程序 mexGetFull 成功执行，其返回值为 0，否则为 1。

举　例：程序 mexGetFull.f 是一个使用函数 mexGetFull 的范例程序，它完成的功能相当简单，仅仅是读取输入参数的内容，并原封不动地赋值给输出参数，其源代码如下：

```
C     mexGetFull.f
C     入口点子例行程序声明及形参类型说明
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      ! MS $ ATTRIBUTES DLLEXPORT :: MEXFUNCTION
      integer nlhs, nrhs
      integer plhs (*), prhs (*)

C     对 MEX 文件中使用的 API 函数进行声明
      integer mxIsDouble, mxIsComplex, mexGetFull
      integer mxGetPr, mxGetPi, mxCreateFull
      character * 32 mxGetName

C     程序中使用变量的声明
      integer i, j
      integer size
      integer pr, pi, m, n, pr_in, pi_in
      real * 8 x (1000), y (1000)
      character * 32 name

C     检查输入和输出变量的个数
```

```
      if (nrhs.eq. 0) then
          call mexErrMsgTxt ('Input arguments required.')
      endif
      if (nlhs .eq. 0) then
          call mexErrMsgTxt ('Output arguments required.')
      endif
      if (nlhs .ne. nrhs) then
          call mexErrMsgTxt ('Number of output arguments
     &                      must equal number of input arguments.')
      endif
C     检查所有输入变量的类型
      do 10 i = 1, nrhs
          if (mxIsDouble (prhs (i)) .ne. 1) then
              call mexErrMsgTxt ('Input arguments
     &                                  must be double type.')
          endif
 10   continue
C     将输入参数分为复数和实数类型进行分别处理
      do 20 j = 1, nrhs
          if (mxIsComplex (prhs (j)) .eq. 1) then
C     复数类型处理:
              name = mxGetName (prhs (j))
              status = mexGetFull (name, m, n, pr_in, pi_in)
              plhs (j) = mxCreateFull (m, n, 1)
              size = m * n
              pr = mxGetPr (plhs (j))
              pi = mxGetPi (plhs (j))
              call mxCopyPtrToReal8 (pr_in, x, size)
              call mxCopyReal8ToPtr (x, pr, size)
              call mxCopyPtrToReal8 (pi_in, y, size)
              call mxCopyReal8ToPtr (y, pi, size)
          else
C     实数类型处理:
              name = mxGetName (prhs (j))
              status = mexGetFull (name, m, n, pr_in, pi_in)
              plhs (j) = mxCreateFull (m, n, 0)
              size = m * n
              pr = mxGetPr (plhs (j))
              call mxCopyPtrToReal8 (pr_in, x, size)
              call mxCopyReal8ToPtr (x, pr, size)
          endif
 20   continue
      return
      end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? a=rand (5);
? b=1-i;
? [c, d] =mexgetfull (a, b)
```

回车后 MATLAB 将显示如下结果

```
c =
    0.9501    0.7621    0.6154    0.4057    0.0579
    0.2311    0.4565    0.7919    0.9355    0.3529
    0.6068    0.0185    0.9218    0.9169    0.8132
    0.4860    0.8214    0.7382    0.4103    0.0099
    0.8913    0.4447    0.1763    0.8936    0.1389
d =
    1.0000- 1.0000i
```

8. mexGetGlobal

功　能：从 MATLAB 的全局变量空间中获取一个 mxArray 结构体的指针。

语　法：integer * 4 function mexGetGlobal (name)
character * (*) name

说　明：函数为一个函数子程序，通过该函数子程序，MEX 文件可以根据名字从 MATLAB 的全局变量空间中获取一个 mxArray 结构体的指针。其输入参数为一个字符串类型的变量或常量，代表一个 mxArray 结构体的名字，若该函数子程序执行成功，将返回一个指向一个 mxArray 结构体的指针，存放在一个整型变量中，否则将返回 0。

举　例：程序 mexgetglobal.f 是一个使用函数 mexGetGlobal 的范例程序，它完成的功能相当简单，仅仅是读入 MATLAB 全局变量空间中的某个变量的内容，并原封不动地赋值给输出参数，程序中我们假设存在一个名为 a 的全局变量，其源代码如下：

```
C     入口点子例行程序声明及形参类型说明
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      ! MS $ ATTRIBUTES DLLEXPORT :: MEXFUNCTION
      integer nlhs, nrhs
      integer plhs (*), prhs (*)

C     对 MEX 文件中使用的 API 函数进行声明
      integer mxIsDouble, mxIsComplex, mexGetGlobal
      integer mxGetPr, mxCreateFull
      integer mxGetM, mxGetN

C     变量声明
      integer size, ptr
      integer pr, m, n, pr _in
      real * 8 x (1000)

C     检查输入和输出参数的个数
      if (nrhs .ne. 0) then
```

```
          call mexErrMsgTxt ('No input arguments required.')
      endif
      if (nlhs .ne. 1) then
          call mexErrMsgTxt ('One output arguments required.')
      endif
C     获取 MATLAB 全局变量 a 的内存指针
      ptr = mexGetGlobal ('a')
      if (ptr .eq. 0) then
          call mexErrMsgTxt ('No such a var.')
      endif
C     获取变量 a 的维数
      m = mxGetM (ptr)
      n = mxGetN (ptr)
      size = m * n
C     检查变量 a 是否为双精度类型
      if (mxIsDouble (ptr) .ne. 1) then
          call mexErrMsgTxt ('Var must be Double type.')
      endif
C     限制变量 a 的大小
      if (size .gt. 1000) then
          call mexErrMsgTxt ('Var is too big.')
      endif
C     检查变量 a 是否实数类型
      if (mxIsComplex (ptr) .eq. 1) then
          call mexErrMsgTxt ('Var must be real.')
      endif
C     为输出参数赋值
      pr_in = mxGetPr (ptr)
      plhs (1) = mxCreateFull (m, n, 0)
      pr = mxGetPr (plhs (1))
      call mxCopyPtrToReal8 (pr_in, x, size)
      call mxCopyReal8ToPtr (x, pr, size)

      return
      end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? a=rand (5);
? global a
? b=mexgetglobal
```

回车后 MATLAB 将显示如下结果

```
b =
    0.9501    0.7621    0.6154    0.4057    0.0579
    0.2311    0.4565    0.7919    0.9355    0.3529
    0.6068    0.0185    0.9218    0.9169    0.8132
```

```
0.4860    0.8214    0.7382    0.4103    0.0099
0.8913    0.4447    0.1763    0.8936    0.1389
```

9. mexGetInf

功　能：获取无穷大的值。

语　法：real * 8 function mexGetInf ()

说　明：函数 mexGetInf 为一个函数子程序，通过该函数子程序，用户可以从返回值中获得自己系统上无穷大的值。Inf 为 IEEE 所规定的一个用来表示无穷大的符号，在下列情况下运算将返回 Inf 作为计算结果：

- 被零除，如 10 / 0
- 数值溢出，如 exp (100000)

举　例：函数子程序 mexGetInf 的使用方法极为简单，不过在使用前必须首先在程序的说明语句段进行声明，如下：

```
real * 8 inf
real * 8 mexGetInf
```

下面就可以直接在程序执行语句中使用了，格式如下：

```
inf = mexGetInf ()
```

10. mexGetMatrix

功　能：从调用者的工作空间复制一个 mxArray 结构体。

语　法：integer * 4 function mexGetMatrix (name)
character * (*) name

说　明：函数 mexGetMatrix 为一个函数子程序，通过该函数子程序，MEX 文件可以根据名字从调用者的工作空间中复制一个 mxArray 结构体。其输入参数为一个字符串类型的变量或常量，代表一个 mxArray 结构体的名字，若该函数子程序执行成功，将返回一个指向一个新分配的 mxArray 结构体的指针，存放在一个整型变量中，否则将返回 0。

举　例：程序 mexgetmatrix.f 是一个使用函数 mexGetMatrix 的范例程序，它完成的功能相当简单，仅仅是读入调用者工作空间中的某个变量的内容，并原封不动地赋值给输出参数，程序中我们假设存在一个名为 a 的变量，其源代码如下：

```
C     mexgetmatrix.f
C     入口点子例行程序声明及形参类型说明
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      ! MS $ ATTRIBUTES DLLEXPORT :: MEXFUNCTION
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     对 MEX 文件中使用的 API 函数进行声明
      integer mxIsDouble, mxIsComplex, mexGetMatrix
      integer mxGetPr, mxGetPi, mxCreateFull
```

```
      integer mxGetString, mxCreateString
      integer mxGetM, mxGetN, mxIsString
C     程序中使用变量的声明
      integer size, ptr
      integer pr, pi, m, n, pr_in, pi_in
      real * 8 x (1000), y (1000)
      character * 1000 string
C     检查输入和输出参数的个数
      if (nrhs .ne. 0) then
          call mexErrMsgTxt ('No input arguments required.')
      endif
      if (nlhs .ne. 1) then
          call mexErrMsgTxt ('One output arguments required.')
      endif
C     假设 MATLAB 工作环境中存在一个名为 a 的变量
      ptr = mexGetMatrix ('a')
      if (ptr .eq. 0) then
          call mexErrMsgTxt ('No such a var.')
      endif
C     获取变量 a 的维数
      m = mxGetM (ptr)
      n = mxGetN (ptr)
      size = m * n
C     检查变量 a 的类型
      if (mxIsDouble (ptr) .eq. 1)       then
C     如果 a 为双精度类型
C     对 a 的大小进行限制
          if (size .gt. 1000) then
              call mexErrMsgTxt ('Var is too big.')
          endif
          if (mxIsComplex (ptr) .eq. 1) then
C             变量 a 为复数类型
              pr_in = mxGetPr (ptr)
              pi_in = mxGetPi (ptr)
              plhs (1) = mxCreateFull (m, n, 1)
              pr = mxGetPr (plhs (1))
              pi = mxGetPi (plhs (1))
              call mxCopyPtrToReal8 (pr_in, x, size)
              call mxCopyReal8ToPtr (x, pr, size)
              call mxCopyPtrToReal8 (pi_in, y, size)
              call mxCopyReal8ToPtr (y, pi, size)
          else
C             变量 a 为实数类型
```

```
                pr_in = mxGetPr (ptr)
                plhs (1) = mxCreateFull (m, n, 0)
                pr = mxGetPr (plhs (1))
                call mxCopyPtrToReal8 (pr_in, x, size)
                call mxCopyReal8ToPtr (x, pr, size)
            endif
        else
C           判断 a 是否为字符串
            if (mxIsString (ptr) .eq. 1) then
                if (m .ne. 1 .or. n .gt. 1000) then
                    call mexErrMsgTxt ('String must be a row
     &                       and less than 1000.')
                endif
                status = mxGetString (ptr, string, size)
                plhs (1) = mxCreateString (string)
                return
            endif
            call mexErrMsgTxt ('Type is not supported.')
        endif

        return
        end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? a=rand (5);
? b=mexgetmatrix
```

回车后 MATLAB 将显示如下结果

```
b =
      0.9501    0.7621    0.6154    0.4057    0.0579
      0.2311    0.4565    0.7919    0.9355    0.3529
      0.6068    0.0185    0.9218    0.9169    0.8132
      0.4860    0.8214    0.7382    0.4103    0.0099
      0.8913    0.4447    0.1763    0.8936    0.1389
```

11. mexGetMatrixPtr

功　能：获取调用者工作空间中某个 mxArray 结构体的指针。

语　法：integer * 4 function mexGetMatrixPtr (name)

character *　(*) name

说　明：函数 mexGetMatrixPtr 为一个函数子程序，通过该函数子程序，MEX 文件可以根据名字从调用者的工作空间中获取某个 mxArray 结构体变量的指针。其输入参数为一个字符串变量，代表了调用者工作空间中的某个 mxArray 结构体类型变量的名字，若该函数子程序执行成功，将返回一个指向名为 name 的 mxArray 结构体的指针，并存放在一个整型变量中，否则将返回 0，结构体 mxArray 既可以为满矩阵，也可以为稀疏矩阵。

通过该函数子程序返回的指针，用户可以在 MEX 文件中利用 MATLAB API 提供的库函数对结构体的数据进行读取和修改。但是这里必须非常注意一点，由函数子程序 mexGetMatrixPtr 所返回的指针所指向的内存区域位于 MATLAB 的工作空间的内部，主要包括了名为 name 的 mxArray 结构体的实数部分、虚数部分以及与稀疏矩阵相关联的各阵列，对于它们的管理主要通过 MATLAB 的内部机制来完成，不允许用户对这些内存段进行任何形式的操作，例如释放和重新分配，否则将导致 MATLAB 系统的立即崩溃。这也是函数子程序 mexGetMatrixPtr 与函数子程序 mexGetMatrix 之间非常重要的一点区别，在函数子程序 mexGetMatrix 中，函数返回的是名为 name 的 mxArray 结构体变量的一个拷贝内存指针，用户在使用完毕后可以对该内存区域进行释放，而不会影响 MATLAB 的运行。

举　例：程序 mexgetmatrixptr.f 是一个使用函数 mexGetMatrixPtr 的范例程序，它完成的功能相当简单，仅仅是读入调用者工作空间中的某个变量的内容，并原封不动地赋值给输出参数，程序中我们假设存在一个名为 a 的变量，其源代码如下：

```
C     mexgetmatrixptr.f
C     入口点子例行程序声明及形参类型说明
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      ! MS$ATTRIBUTES DLLEXPORT :: MEXFUNCTION
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
C     对 MEX 文件中使用的 API 函数进行声明
      integer mxIsDouble, mxIsComplex, mexGetMatrixPtr
      integer mxGetPr, mxGetPi, mxCreateFull
      integer mxGetString, mxCreateString
      integer mxGetM, mxGetN, mxIsString
C     程序中使用变量的声明
      integer size, ptr
      integer pr, pi, m, n, pr_in, pi_in
      real*8 x (1000), y (1000)
      character*1000 string
C     检查输入和输出参数的个数
      if (nrhs .ne. 0) then
         call mexErrMsgTxt ('No input arguments required.')
      endif
      if (nlhs .ne. 1) then
         call mexErrMsgTxt ('One output arguments required.')
      endif
C     假设 MATLAB 工作环境中存在一个名为 a 的变量
      ptr = mexGetMatrixPtr ('a')
      if (ptr .eq. 0) then
```

```
            call mexErrMsgTxt ('No such a var.')
        endif
C       获取变量 a 的维数
        m = mxGetM (ptr)
        n = mxGetN (ptr)
        size = m * n
C       检查变量 a 的类型
        if (mxIsDouble (ptr) .eq. 1) then
C       如果 a 为双精度类型
C           对 a 的大小进行限制
            if (size .gt. 1000) then
                call mexErrMsgTxt ('Var is too big.')
            endif
            if (mxIsComplex (ptr) .eq. 1) then
C               变量 a 为复数类型
                pr_in = mxGetPr (ptr)
                pi_in = mxGetPi (ptr)
                plhs (1) = mxCreateFull (m, n, 1)
                pr = mxGetPr (plhs (1))
                pi = mxGetPi (plhs (1))
                call mxCopyPtrToReal8 (pr_in, x, size)
                call mxCopyReal8ToPtr (x, pr, size)
                call mxCopyPtrToReal8 (pi_in, y, size)
                call mxCopyReal8ToPtr (y, pi, size)
            else
C               变量 a 为实数类型
                pr_in = mxGetPr (ptr)
                plhs (1) = mxCreateFull (m, n, 0)
                pr = mxGetPr (plhs (1))
                call mxCopyPtrToReal8 (pr_in, x, size)
                call mxCopyReal8ToPtr (x, pr, size)
            endif
        else
C           判断 a 是否为字符串
            if (mxIsString (ptr) .eq. 1) then
                if (m .ne. 1 .or. n .gt. 1000) then
                    call mexErrMsgTxt ('String must be a row
     &                      and less than 1000.')
                endif
                status = mxGetString (ptr, string, size)
                plhs (1) = mxCreateString (string)
                return
            endif
```

```
call mexErrMsgTxt ('Type is not supported.')
endif

return
end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? a=rand (5);
? b=mexgetmatrix
```

回车后 MATLAB 将显示如下结果

```
b =
    0.9501    0.7621    0.6154    0.4057    0.0579
    0.2311    0.4565    0.7919    0.9355    0.3529
    0.6068    0.0185    0.9218    0.9169    0.8132
    0.4860    0.8214    0.7382    0.4103    0.0099
    0.8913    0.4447    0.1763    0.8936    0.1389
```

12. mexGetNaN

功　能：用于获取 MATLAB 内部变量 NaN 的值。

语　法：real * 8 function mexGetNan ()

说　明：函数 mexGetNaN 为一个函数子程序，通过该函数子程序，用户可以获得 MATLAB 系统中内部变量 NaN 的值。NaN 为 IEEE 所规定的一个用来表示数学运算中无法用数字表示的量的符号，其含义为 Not-a-Number，在下列情况下，运算返回 NaN 作为运算结果：

- 0.0 / 0.0
- Inf - Inf

举　例：函数子程序 mexGetNaN 的使用方法极为简单，不过在使用前必须首先在程序的说明语句段进行声明，如下：

```
real * 8 NaN
real * 8 mexGetNaN
```

下面就可以直接在程序执行语句中使用了，格式如下：

```
NaN = mexGetNaN ()
```

13. mexIsFinite

功　能：判断一个数量是否为一个有限值。

语　法：integer * 4 function mexIsFinite (value)

real * 8 value

说　明：函数 mexIsFinite 为一个函数子程序，通过该函数子程序，用户可以判断一个数量是否为有限值，一般情况下，如果一个数量既不为 Inf 也不为 NaN，那么它一定为有限值。函数子程序 mexIsFinite 的输入参数为一个双精度浮点类型的数值变量 value，当 value 为一个有限值时，返回值为 1，否则为 0。

举　例：函数子程序 mexIsFinite 的使用方法极为简单，不过在使用前必须首先在程

序的说明语句段进行声明，如下：

integer A

integer mexIsFinite

下面就可以直接在程序执行语句中使用了，格式如下：

A = mexGetNaN (B)

14. mexIsInf

功　能：判断一个数值是否为无穷大 Inf。

语　法：integer * 4 function mexIsInf (value)

real * 8 value

说　明：函数 mexIsInf 为一个函数子程序，通过该函数子程序，用户可以判断一个数量是否为无穷大。其输入参数为一个双精度浮点类型的数值变量 value，当 value 为 Inf 时，函数返回值为 1，否则为 0。有关无穷大 Inf 的定义请读者参见函数 mexGetInf 的说明。

举　例：函数子程序 mexIsInf 的使用方法极为简单，不过在使用前必须首先在程序的说明语句段进行声明，如下：

integer Inf

integer mexIsInf

下面就可以直接在程序执行语句中使用了，格式如下：

Inf = mexIsInf ()

15. mexIsNaN

功　能：判断一个数值是否为 NaN。

语　法：integer * 4 function mexIsNaN (value)

real * 8 value

说　明：函数 mexIsNaN 为一个函数子程序，通过该函数子程序，用户可以判断一个数量是否为 NaN。其输入参数为一个双精度浮点类型的数值变量 value，当 value 为 NaN 时，函数返回值为 1，否则为 0。有关 NaN 的定义请读者参见函数 mexGetNaN 的说明。

举　例：函数子程序 mexIsNaN 的使用方法极为简单，不过在使用前必须首先在程序的说明语句段进行声明，如下：

integer NaN

integer mexIsNaN

下面就可以直接在程序执行语句中使用了，格式如下：

NaN = mexIsNaN ()

16. mexPrintf

功　能：输出信息。

语　法：subroutine mexPrintf (message)

character *　(*) message

说　明： 函数 mexPrintf 为一个子例行程序，用于向屏幕上输出一定的信息，在 MEX 文件中可以直接使用 call 关键字调用，其输入参数为一个字符串变量，它包含了用户希望输出的信息。这里必须非常注意的一点是符号%对于函数 mexPrintf 来说具有特殊的含义，如果用户希望在输出的信息中包含符号%，则必须书写为%%，否则将会产生意想不到的结果。

举　例： call mexPrintf ('Please input the matrix:')

17. mexPutFull

功　能： 用于向调用者的工作空间输出一个存储类型为满的 mxArray 结构体。

语　法： integer * 4 function mexPutFull (name, m, n, pr, pi)

integer * 4 m, n, pr, pi

character *　(*) name

说　明： 函数 mexPutFull 为一个函数子程序，通过该函数子程序，用户可以方便地向 MEX 文件的调用者的工作空间输出一个存储类型为满的 mxArray 结构体，其五个输入参数的类型和含义分别如下：

- name 为一个字符串变量，包含了输出的 mxArray 结构体的名字；
- m 为一个整型变量，包含了输出的 mxArray 结构体的行向量的个数；
- n 为一个整型变量，包含了输出的 mxArray 结构体的列向量的个数；
- pr 为一个整型变量，包含了指向输出的 mxArray 结构体的实数数据部分的指针；
- pi 为一个整型变量，包含了指向输出的 mxArray 结构体的虚数数据部分的指针。

如果该函数执行成功，其返回值为 0，否则为 1。如果在调用者的工作空间中已经存在名为 name 的 mxArray 结构体，那么旧的结构体将为新的结构体所覆盖。

举　例： 程序 mexputfull. f 是一个使用函数 mexPutFull 的范例程序，它完成的功能相当简单，首先对输入的参数进行类型判断，并分类，若输入参数为复数类型，则将输入参数的实部和虚部交换，若输入参数为实数类型，则将所有的数据取绝对值，完成这些工作后，以相同的名字将操作结果输出到调用者的工作空间，若输入的参数类型不为双精度类型，则报错。程序的源代码如下：

```
C     mexputfull. f
C     入口点子例行程序声明及形参类型说明
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      ! MS $ ATTRIBUTES DLLEXPORT :: MEXFUNCTION
      integer nlhs, nrhs
      integer plhs (*), prhs (*)
```

```
C      对 MEX 文件中使用的 API 函数进行声明
       integer mxIsDouble, mxIsComplex, mexGetFull, mexPutFull
       character * 32 mxGetName
C      变量声明
       integer i, j, typenum (10), status
       integer pr, pi, m, n, size
       real * 8 x (1000)
       character * 32 name
C      对输入和输出参数个数的检查
       if (nrhs .eq. 0) then
           call mexErrMsgTxt ('At least one input argument required.')
       endif
       if (nrhs .gt. 10) then
           call mexErrMsgTxt ('Input argument must be less than 10.')
       endif
       if (nlhs .ne. 0) then
           call mexErrMsgTxt ('No output argument required.')
       endif
C      对输入参数类型的判断
       do 10 i = 1, nrhs
           if (mxIsDouble (prhs (i)) .eq. 1) then
               if (mxIsComplex (prhs (i)) .eq. 1) then
                   typenum (i) = 1
               else
                   typenum (i) = 2
               endif
           endif
 10    continue
C      处理过程
       do 20 j = 1, nrhs
           select case (typenum (j))
C              当输入参数为复数类型时：将实部与虚部进行交换
               case (1)
                   name = mxGetName (prhs (j))
                   if (mexGetFull (name, m, n, pr, pi) .ne. 0) then
                       call mexErrMsgTxt ('Reading input
       &                                     argument occurs error.')
                   endif
                   size = m * n
                 if (size .gt. 1000) then
                     call mexErrMsgTxt ('Input argument is too big.')
                 endif
                 status = mexPutFull (name, m, n, pi, pr)
```

```
C               当输入参数为实型时：取绝对值
                case (2)
                    name = mxGetName (prhs (j))
                    if (mexGetFull (name, m, n, pr_in, pi) .ne. 0) then
                        call mexErrMsgTxt ('Reading input
     &                                          argument occurs error.')
                    endif
                    size = m * n
                    if (size .gt. 1000) then
                        call mexErrMsgTxt ('Input argument is too big.')
                    endif
                    call mxCopyPtrToReal8 (pr, x, size)
                    x = ABS (x)
                    call mxCopyReal8ToPtr (x, pr, size)
                    status = mexPutFull (name, m, n, pr, pi)

            END SELECT
   20   continue

        return
        end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? a=1-2i
a =
    1.0000 - 2.0000i
? b=-1
b =
    -1
? mexputfull (a, b)
```

回车后，键入以下命令可得

```
? a
a =
    -2.0000 + 1.0000i
? b
b =
    1
```

可见变量 a 和 b 已经按要求进行了改变。

18. mexPutMatrix

功　能：向调用者的工作空间输出一个 mxArray 结构体。

语　法：integer * 4 function mexPutMatrix (mp)

integer * 4 mp

说　明：函数 mexPutMatrix 为一个函数子程序，它与函数 mexPutFull 所实现的功能一致，不过实现的方式不同。函数子程序 mexPutMatrix 仅有一个类型为整型输入参数 mp，它包含了希望输出到调用者工作空间中的 mxArray 结

构体的内存指针。如果该函数子程序成功执行，其返回值为 0，否则为 1。当调用者的工作空间中存在与输出的 mxArray 结构体同名的 mxArray 结构体时，原有的 mxArray 结构体将被覆盖。

举　例： 程序 mexputmatrix.f 是一个使用函数 mexPutMatrix 的范例程序，它完成的功能相当简单，首先对输入的参数进行类型判断，并分类，若输入参数为复数类型，则将输入参数的实部和虚部交换；若输入参数为实数类型，则将所有的数据取绝对值；若输入的参数为字符串类型，则保持原样不动，完成这些工作后，以相同的名字将操作结果输出到调用者的工作空间。程序的源代码如下：

```
C     mexputmatrix.f
C     入口点子例行程序声明及形参类型说明
      subroutine mexFunction (nlhs, plhs, nrhs, prhs)
      ! MS $ ATTRIBUTES DLLEXPORT :: MEXFUNCTION
      integer nlhs, nrhs
      integer plhs (*), prhs (*)

C     对 MEX 文件中使用的 API 函数进行声明
      integer mxIsDouble, mxIsString, mxIsComplex
      integer mxGetM, mxGetN, mxGetPr, mxGetPi
      integer mxGetString, mxCreateString
      character * 32 mxGetName

C     程序中使用变量的声明
      integer typenum (100), i, j, output (100)
      integer ptr _ inx, ptr _ outx, ptr _ iny, ptr _ outy, m, n, size
      real * 8 x (1000), y (1000)
      character * 100 string
      character * 32 name (100)

C     检查输入和输出参数的个数
      if (nrhs .eq. 0) then
          call mexErrMsgTxt ('Input arguments required.')
      endif

      if (nrhs .gt. 100) then
          call mexErrMsgTxt ('Input arguments must be less than 100.')
      endif

      if (nlhs .ne. 0) then
          call mexErrMsgTxt ('No output arguments required.')
      endif

C     检查所有输入参数的类型，并记录
      do 10 i = 1, nrhs
          if (mxIsDouble (prhs (i)) .eq. 1) then
              typenum (i) = 1
          endif
          if (mxIsComplex (prhs (i)) .eq. 1) then
```

```
          typenum (i) = 2
          endif
          if (mxIsString (prhs (i)) .eq. 1) then
              typenum (i) = 3
          endif
          name (i) = mxGetName (prhs (i))
 10   continue
C     对输出参数进行与输入参数相应的类型设置并赋值
      do 20 j = 1, nrhs
          select case (typenum (j))
C     当输入参数为双精度类型的实数时:
              case (1)
                  m = mxGetM (prhs (j))
                  n = mxGetN (prhs (j))
                  size = m * n
                  if (size .gt. 1000) then
                      call mexErrMsgTxt ('Input argument is too big.')
                  endif
                  output (j) = mxCreateFull (m, n, 0)
                  ptr_inx = mxGetPr (prhs (j))
                  ptr_outx = mxGetPr (output (j))
                  call mxCopyPtrToReal8 (ptr_inx, x, size)
                  x = ABS (x)
                  call mxCopyReal8ToPtr (x, ptr_outx, size)
                  call mxSetName (output (j), name (j))
                  status = mexPutMatrix (output (j))
C     当输入参数为双精度类型的复数时:
              case (2)
                  m = mxGetM (prhs (j))
                  n = mxGetN (prhs (j))
                  size = m * n
                  if (size .gt. 1000) then
                      call mexErrMsgTxt ('Input argument is too big.')
                  endif
                  output (j) = mxCreateFull (m, n, 1)
                  ptr_inx = mxGetPr (prhs (j))
                  ptr_outx = mxGetPr (output (j))
                  ptr_iny = mxGetPi (prhs (j))
                  ptr_outy = mxGetPi (output (j))
                  call mxCopyPtrToReal8 (ptr_inx, x, size)
                  call mxCopyReal8ToPtr (x, ptr_outy, size)
                  call mxCopyPtrToReal8 (ptr_iny, y, size)
                  call mxCopyReal8ToPtr (y, ptr_outx, size)
```

```
                  call mxSetName (output (j), name (j))
                  status = mexPutMatrix (output (j))
C     当输入参数为字符串类型时:
               case (3)
                  m = mxGetM (prhs (j))
                  n = mxGetN (prhs (j))
                  size = m * n


                  if (m .ne. 1) then
                     call mexErrMsgTxt ('String must be a row vector')
                  elseif (n .gt. 1000) then
                     call mexErrMsgTxt ('String is too long.')
                  endif
                  if (mxGetString (prhs (j), string, size) .ne. 0) then
                     call mexErrMsgTxt ('error')
                  endif
                  output (j) = mxCreateString (string)
                  call mxSetName (output (j), name (j))
                  status = mexPutMatrix (output (j))
            END SELECT
 20   continue
      return
      end
```

对该程序编译后，在 MATLAB 命令提示符下键入以下命令

```
? a=1-2i
a =
   1.0000 - 2.0000i
? b=-1
b =
   -1
? c='abcd'
c =
abcd
? mexputmatrix (a, b, c)
```

回车后键入以下命令可得

```
? a
a =
   -2.0000 + 1.0000i
? b
b =
   1
? c
c =
```

abcd

可见变量 a、b 和 c 已经按要求进行了改变。

19. mexSetTrapFlag

功　能：用于设置调用函数 mexCallMATLAB 发生错误时控制流的走向。

语　法：subroutine mexSetTrapFlag (trap _ flag)

integer * 4 trap _ flag

说　明：函数 mexSetTrapFlag 为一个子例行程序，在 MEX 文件中可以直接使用 call 关键字调用，其输入参数为一个整型变量 trap _ flag，目前其合法的取值只能为 0 和 1，分别代表了不同的含义，其中 0 代表在调用函数 mexCallMATLAB 发生错误时将控制返回给 MATLAB，1 代表在调用函数 mexCallMATLAB 发生错误时将控制返回给 MEX 文件。

如果在用户的 MEX 文件中，没有使用函数 mexSetTrapFlag 进行控制流的走向设置，那么在调用函数 mexCallMATLAB 发生错误时系统立即将控制返回到 MATLAB 下；如果使用函数 mexSetTrapFlag 进行了设置但是标志为 0，则在调用函数 mexCallMATLAB 发生错误时系统同样将控制返回到 MATLAB 系统；只有当使用函数 mexSetTrapFlag 进行了设置并且标志为 1 时，调用函数 mexCallMATLAB 发生错误时系统才会将控制返回到当前的 MEX 文件，并且位于调用函数 mexCallMATLAB 语句的下面一行，这时用户可以进行自己的错误处理。

举　例：函数 mexCallMATLAB 的使用非常方便，只需按如下格式调用即可：

call mexCallMATLAB (0)

或 call mexCallMATLAB (1)

或 call mexCallMATLAB (trap _ flag)

其中 trap _ flag 为一个 integer 类型的变量，取值仅可以为 0 和 1。

4.7 FORTRAN 语言 mx-函数

4.7.1 FORTRAN 语言 mx-函数的声明

在 MATLAB 应用程序接口函数库中，总共提供了 39 个 FORTRAN 语言的 mx-函数，它们的声明分别如下：

```
integer * 4 function mxCalloc (n, size)
subroutine mxCopyCharacterToPtr (y, px, n)
subroutine mxCopyComplex16ToPtr (y, pr, pi, n)
subroutine mxCopyInteger4ToPtr (y, px, n)
subroutine mxCopyPtrToCharacter (px, y, n)
subroutine mxCopyPtrToComplex16 (pr, pi, y, n)
subroutine mxCopyPtrToInteger4 (px, y, n)
subroutine mxCopyPtrToPtrArray (px, y, n)
```

```
subroutine mxCopyPtrToReal8 (px, y, n)
subroutine mxCopyReal8ToPtr (y, px, n)
integer * 4 function mxCreateFull (m, n, ComplexFlag)
integer * 4 function mxCreateSparse (m, n, nzmax, ComplexFlag)
integer * 4 function mxCreateString (str)
subroutine mxFree (ptr)
subroutine mxFreeMatrix (pm)
integer * 4 function mxGetIr (pm)
integer * 4 function mxGetJc (pm)
integer * 4 function mxGetM (pm)
integer * 4 function mxGetN (pm)
character * 32 function mxGetName (pm)
integer * 4 function mxGetNzmax (pm)
integer * 4 function mxGetPi (pm)
integer * 4 function mxGetPr (pm)
real * 8 function mxGetScalar (pm)
integer * 4 function mxGetString (pm, str, strlen)
integer * 4 function mxIsComplex (pm)
integer * 4 function mxIsDouble (pm)
integer * 4 function mxIsFull (pm)
integer * 4 function mxIsNumeric (pm)
integer * 4 function mxIsSparse (pm)
integer * 4 function mxIsString (pm)
subroutine mxSetIr (pm, ir)
subroutine mxSetJc (pm, jc)
subroutine mxSetM (pm, m)
subroutine mxSetN (pm, n)
subroutine mxSetName (pm, name)
subroutine mxSetNzmax (pm, nzmax)
subroutine mxSetPi (pm, pi)
subroutine mxSetPr (pm, pr)
```

4.7.2 FORTRAN 语言 mx-函数的使用说明

1. mxCalloc

功　能：动态分配内存。

语　法：integer * 4 function mxCalloc (n, size)

　　　　integer * 4 n, size

说　明：函数 mxCalloc 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxCalloc，用户可以在 MATLAB 接口应用程序中方便地完成动态内存分配任务。形式参数 n 指明了所分配的内存中存放元素的数量，而形式参数 size 则说明了每一个元素所占用的字节数，所以实际分配内存的字节数为 n *

size。如果函数执行成功，将返回一个指向所分配内存起始字节地址的指针；如果函数执行失败，在 MEX 文件中，函数将终止整个程序的运行，并将控制返回到 MATLAB 命令提示符下，而在其他的 MATLAB 接口应用程序中，函数将返回 0，并不终止应用程序的执行。造成两种不同失败返回结果的主要原因是内存的管理方式不同。在 MEX 文件中，MATLAB 的自动内存管理机制会对所有的由函数 mxCalloc 分配的内存进行记录和管理，在 MEX 文件结束执行时，对文件中所分配的内存进行自动释放，而无须用户进行显式的释放，不过在使用完毕一段内存后，使用函数 mxFree 进行释放，是一种良好的编程习惯。在 MEX 文件中，函数 mxCalloc 的执行，将自动完成下面三个任务：

- 分配足够的堆内存；
- 将内存中所有 n 各元素初始化为 0；
- 在 MATLAB 的内存自动管理机制中对分配的内存进行注册，以便于在程序结束时进行自动释放。

而在其他的 MATLAB 接口应用程序，如 MAT 文件应用程序和 MATLAB 引擎应用程序中，MATLAB 的自动内存管理机制不会发生作用，而是依靠语言本身的内存管理机制。

举　例：参见 mex-函数 mexAtExit 的范例程序。

2. mxCopyCharacterToPtr

功　能：将一个 FORTRAN 语言字符串的内容复制到某个指定的阵列中。

语　法：subroutine mxCopyCharacterToPtr (y, px, n)

```
character * (*) y
integer * 4 px, n
```

说　明：函数 mxCopyCharacterToPtr 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyCharacterToPtr，用户可以将一个 FORTRAN 语言字符串的内容复制到某个指定的阵列中，函数的三个形式参数的含义分别如下：

- y 为一个 FORTRAN 语言的字符串；
- px 为一个指向某个阵列的指针；
- n 为希望从 y 中复制到 px 指向的阵列中的字符串的长度。

3. mxCopyComplex16ToPtr

功　能：将一个 FORTRAN 语言复数类型的数组的数据复制到某个指定的复数类型的阵列中。

语　法：subroutine mxCopyComplex16ToPtr (y, pr, pi, n)

```
complex * 16 y (n)
integer * 4 pr, pi, n
```

说　明：函数 mxCopyComplex16ToPtr 为一个 FORTRAN 语言的子例行程序，在

MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyComplex16ToPtr，用户可以将一个 FORTRAN 语言复数数组的内容复制到某个指定的复数类型的阵列中，函数的四个形式参数的含义分别如下：

- y 为一个 FORTRAN 语言复数数组；
- pr 为指向复数阵列的实数部分数据的指针；
- pi 为指向复数阵列的虚数部分数据的指针；
- n 为希望复制的数组元素的个数。

举 例：参见 4.2.6 节的范例程序。

4. mxCopyInteger4ToPtr

功 能：将一个 FORTRAN 语言整数类型的数组的数据复制到某个指定的稀疏阵列的 ir 或 jc 数组中。

语 法：subroutine mxCopyInteger4ToPtr（y，px，n）

integer * 4 y（n）

integer * 4 px，n

说 明：函数 mxCopyInteger4ToPtr 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyInteger4ToPtr，用户可以将一个 FORTRAN 语言整数数组的内容复制到某个指定的稀疏矩阵的 ir 或 jc 数组中，函数的三个形式参数的含义分别如下：

- y 为一个 FORTRAN 语言的整数类型数组；
- px 为指向稀疏矩阵的 ir 或 jc 数组数据的指针；
- n 为希望复制的元素的数量。

5. mxCopyPtrToCharacter

功 能：将一个字符串类型阵列所包含的字符串复制到一个 FORTRAN 语言的字符串变量中。

语 法：subroutine mxCopyPtrToCharacter（px，y，n）

character * （*）y

integer * 4 px，n

说 明：函数 mxCopyPtrToCharacter 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyPtrToCharacter，用户可以将一个字符串类型阵列所包含的字符串复制到一个 FORTRAN 语言的字符串变量中，函数的三个形式参数的含义分别如下：

- px 为一个指向字符串类型阵列的指针；
- y 为一个用于存放从阵列中读取出的字符串的 FORTRAN 语言的字符串变量；

• n 为希望读取出的字符串的长度。

该函数与函数 mxCopyCharacterToPtr 互为逆函数，它们所完成的功能正好相反。

6. mxCopyPtrToComplex16

功　能：将一个复数类型的阵列所包含的数据复制到一个 FORTRAN 语言的复数类型的数组中。

语　法：subroutine mxCopyPtrToComplex16 (pr, pi, y, n)

complex * 16 y (n)

integer * 4 pr, pi, n

说　明：函数 mxCopyPtrToComplex16 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyPtrToComplex16，用户可以将一个复数类型的阵列所包含的数据复制到一个 FORTRAN 语言的复数类型的数组中，函数的四个形式参数的含义分别如下：

• pr 为指向复数类型阵列的实数部分数据的指针；
• pi 为指向复数类型阵列的虚数部分数据的指针；
• y 为一个 FORTRAN 语言的复数类型的数组；
• n 为希望复制的元素的个数。

该函数与函数 mxCopyComplex16ToPtr 互为逆函数，它们所完成的功能正好相反。

举　例：参见 4.2.6 节的范例程序。

7. mxCopyPtrToInteger4

功　能：将一个稀疏矩阵的 ir 或 jc 数组的数据复制到一个 FORTRAN 语言的整数类型的数组中。

语　法：subroutine mxCopyPtrToInteger4 (px, y, n)

integer * 4 y (n)

integer * 4 px, n

说　明：函数 mxCopyPtrToInteger4 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyPtrToInteger4，用户可以将一个稀疏矩阵的 ir 或 jc 数组的数据复制到一个 FORTRAN 语言的整数类型的数组中，函数的三个形式参数的含义分别如下：

• px 为指向稀疏矩阵的 ir 或 jc 数组数据的指针；
• y 为一个 FORTRAN 语言的整数类型数组；
• n 为希望复制的元素的个数。

该函数与函数 mxCopyInteger4ToPtr 互为逆函数，它们所完成的功能正好相反。

8. mxCopyPtrToPtrArray

功　能：复制一个指针到一个 FORTRAN 语言的整数类型的数组中。

语　法：subroutine mxCopyPtrToPtrArray (px, y, n)

```
integer * 4 y (n)
integer * 4 px, n
```

说　明：函数 mxCopyPtrToPtrArray 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyPtrToPtrArray，用户可以将指向稀疏矩阵中的 ir 和 jc 数组的指针，复制一个 FORTRAN 语言的整数类型的数组中。

9. mxCopyPtrToReal8

功　能：将某个阵列的实数或虚数部分的数据复制到一个 FORTRAN 语言的实数类型的数组中。

语　法：subroutine mxCopyPtrToReal8 (px, y, n)

```
real * 8 y (n)
integer * 4 px, n
```

说　明：函数 mxCopyPtrToReal8 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyPtrToReal8，用户可以将某个阵列的实数或虚数部分的数据复制到一个 FORTRAN 语言的实数类型的数组中，函数的三个形式参数的含义分别如下：

- px 为指向某个阵列的实数或虚数部分的数据的指针；
- y 为一个 FORTRAN 语言的实数类型数组；
- n 为希望复制的元素的个数。

举　例：参见 4.2.2 节的范例程序。

10. mxCopyReal8ToPtr

功　能：将一个 FORTRAN 语言的实数类型数组中的数据复制到某个阵列的实数部分或虚数部分中。

语　法：subroutine mxCopyReal8ToPtr (y, px, n)

```
real * 8 y (n)
integer * 4 px, n
```

说　明：函数 mxCopyReal8ToPtr 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxCopyReal8ToPtr，用户可以将一个 FORTRAN 语言的实数类型数组中的数据复制到某个阵列的实数部分或虚数部分中，函数的三个形式参数的含义分别如下：

- y 为一个 FORTRAN 语言的实数类型数组；

- px 为指向某个阵列的实数或虚数部分的数据的指针；
- n 为希望复制的元素的个数。

该函数与函数 mxCopyPtrToReal8 互为逆函数，它们所完成的功能正好相反。

举 例：参见 4.2.2 节的范例程序。

11. mxCreateFull

功 能：创建一个二维的存储类型为满的未赋值的阵列。

语 法：integer * 4 function mxCreateFull (m, n, ComplexFlag)
integer * 4 m, n, ComplexFlag

说 明：函数 mxCreateFull 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxCreateFull，用户可以方便地创建一个二维的存储类型为满的未赋值的阵列。函数的三个形式参数的含义分别如下：

- m 为所创建的阵列的行数；
- n 为所创建的阵列的列数；
- ComplexFlag 为所创建的阵列的类型，当 ComplexFlag = 1 时，创建复数类型的阵列；当 ComplexFlag = 0 时，创建实数类型的阵列。

如果函数执行成功，将返回一个指向所创建阵列的指针；否则返回 0。

举 例：参见 4.2.2 节的范例程序。

12. mxCreateSparse

功 能：创建一个未赋值的稀疏矩阵。

语 法：integer * 4 function mxCreateSparse (m, n, nzmax, ComplexFlag)
integer * 4 m, n, nzmax, ComplexFlag

说 明：函数 mxCreateSparse 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxCreateSparse，用户可以方便的创建一个未赋值的稀疏矩阵。函数的四个形式参数的含义分别如下：

- m 为所创建的稀疏矩阵的行数；
- n 为所创建的稀疏矩阵的列数；
- nzmax 为稀疏矩阵中可以存放的非零元素的最大个数；
- ComplexFlag 为所创建的阵列的类型，当 ComplexFlag = 1 时，创建复数类型的阵列；当 ComplexFlag = 0 时，创建实数类型的阵列。

如果函数执行成功，将返回一个指向所创建稀疏矩阵的指针；否则返回 0。

举 例：参见 4.2.7 节的范例程序。

13. mxCreateString

功 能：创建一个大小为 1×n 的字符串类型的阵列，并通过形式参数赋值。

语　法：integer * 4 function mxCreateString (str)
　　character * (*) str

说　明：函数 mxCreateString 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mx-CreateString，用户可以方便地创建一个大小为 1×n 的字符串类型的阵列，并通过字符串类型的形式参数 str 进行赋值。

举　例：参见 4.2.1 节的范例程序。

14. mxFree

功　能：释放动态分配的内存。

语　法：subroutine mxFree (ptr)
　　integer * 4 ptr

说　明：函数 mxFree 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxFree，用户可以方便地释放由函数 mxCalloc 动态分配的内存。

举　例：参见 4.2.3 节的范例程序。

15. mxFreeMatrix

功　能：释放由函数 mxCreateSparse 和函数 mxCreateFull 在创建阵列时动态分配的内存。

语　法：subroutine mxFreeMatrix (pm)
　　integer * 4 pm

说　明：函数 mxFreeMatrix 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxFreeMatrix，用户可以方便地释放由函数 mxCreateSparse 和函数 mxCreateFull 在创建阵列时动态分配的内存。

举　例：参见 4.2.3 节的范例程序。

16. mxGetIr

功　能：获得稀疏矩阵的 ir 数组的内容。

语　法：integer * 4 function mxGetIr (pm)
　　integer * 4 pm

说　明：函数 mxGetIr 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetIr，用户可以方便地获得由形式参数 pm 指向的稀疏矩阵的 ir 数组的内容。如果函数执行成功，将返回指向 ir 数组第一个元素的指针，如果函数执行失败，将返回 0。一般情况下，函数失败的原因主要有：

- pm 指向的阵列不是稀疏矩阵；
- 前面对函数 mxCreateSparse 的调用失败。

有关稀疏矩阵的 ir 数组的说明，请读者自行参阅相关章节。

举 例：参见 4.2.7 节的范例程序。

17. mxGetJc

功 能：获得稀疏矩阵的 jc 数组的内容。

语 法：integer * 4 function mxGetJc (pm)

integer * 4 pm

说 明：函数 mxGetJc 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetJc，用户可以方便地获得由形式参数 pm 指向的稀疏矩阵的 jc 数组的内容。如果函数执行成功，将返回指向 jc 数组第一个元素的指针，如果函数执行失败，将返回 0。一般情况下，函数失败的原因主要有：

- pm 指向的阵列不是稀疏矩阵；
- 前面对函数 mxCreateSparse 的调用失败。

有关稀疏矩阵的 jc 数组的说明，请读者自行参阅相关章节。

举 例：参见 4.2.7 节的范例程序。

18. mxGetM

功 能：获得阵列的行数。

语 法：integer * 4 function mxGetM (pm)

integer * 4 pm

说 明：函数 mxGetM 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetM，用户可以方便地获得由形式参数 pm 指定的阵列的行数。

举 例：参见 4.2.1 节的范例程序。

19. mxGetN

功 能：获得阵列的列数。

语 法：integer * 4 function mxGetN (pm)

integer * 4 pm

说 明：函数 mxGetN 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetN，用户可以方便地获得由形式参数 pm 指定的阵列的列数。

举 例：参见 4.2.1 节的范例程序。

20. mxGetName

功 能：获得阵列的名字。

语 法：character * 32 function mxGetName (pm)

integer * 4 pm

说　明：函数 mxGetName 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetName，用户可以方便地获得由形式参数 pm 指定的阵列的名字。

举　例：参见 mex-函数 mexGetFull 的范例程序。

21. mxGetNzmax

功　能：获得稀疏矩阵所能存放的最多的非零元素的个数。

语　法：integer * 4 function mxGetNzmax (pm)
　　integer * 4 pm

说　明：函数 mxGetNzmax 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetNzmax，用户可以方便地获得由形式参数 pm 指定的稀疏矩阵所能存放的最多的非零元素的个数。

22. mxGetPi

功　能：获得阵列的虚数部分的数据。

语　法：integer * 4 function mxGetPi (pm)
　　integer * 4 pm

说　明：函数 mxGetPi 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetPi，用户可以方便地获得由形式参数 pm 指定的阵列的虚数部分的数据。如果函数执行成功，函数将返回指向阵列虚数部分数据第一个元素的指针，否则返回 0。

举　例：参见 4.2.6 节的范例程序。

23. mxGetPr

功　能：获得阵列的实数部分的数据。

语　法：integer * 4 function mxGetPr (pm)
　　integer * 4 pm

说　明：函数 mxGetPr 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetPr，用户可以方便地获得由形式参数 pm 指定的阵列的实数部分的数据。如果函数执行成功，函数将返回指向阵列实数部分数据第一个元素的指针，否则返回 0。

举　例：参见 4.2.6 节的范例程序。

24. mxGetScalar

功　能：获得阵列实数部分数据的第一个元素的值。

语　法：real * 8 function mxGetScalar (pm)

integer * 4 pm

说　明：函数 mxGetScalar 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetScalar，用户可以方便地获得由形式参数 pm 指定的阵列实数部分数据的第一个元素的值。通常该函数用于提取仅包含一个数量的阵列的内容；如果 pm 所指向的阵列包含多个元素或者为多维，则函数提取实数部分数据的第一个元素的值。

25. mxGetString

功　能：从一个字符串类型阵列中读取内容，并存放到一个 FORTRAN 语言的字符串变量中。

语　法：integer * 4 function mxGetString (pm, str, strlen)

integer * 4 pm, strlen

character * (*) str

说　明：函数 mxGetString 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxGetString，用户可以方便地从一个字符串阵列中读取内容，并存放到一个 FORTRAN 语言的字符串变量中。函数的三个形式参数的含义分别如下：

- pm 为指向字符串类型阵列的指针；
- str 为一个 FORTRAN 语言的字符串变量；
- strlen 为希望读取的字符串的长度。

如果函数执行成功，将返回 0，否则返回 1。

举　例：参见 4.2.1 节的范例程序。

26. mxIsComplex

功　能：判断一个阵列是否为复数类型。

语　法：integer * 4 function mxIsComplex (pm)

integer * 4 pm

说　明：函数 mxIsComplex 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxIsComplex，用户可以方便地判断由形式参数 pm 指定的阵列是否为复数类型，如果是，则函数返回 1，否则返回 0。

举　例：参见 4.2.6 节的范例程序。

27. mxIsDouble

功　能：判断一个阵列是否为双精度类型。

语　法：integer * 4 function mxIsDouble (pm)

integer * 4 pm

说　明：函数 mxIsDouble 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxIsDouble，用户可以方便地判断由形式参数 pm 指定的阵列是否为双精度类型，如果是，则函数返回 1，否则返回 0。

举　例：参见 4.2.7 节的范例程序。

28. mxIsFull

功　能：判断一个阵列是否为满存储类型的阵列。

语　法：integer * 4 function mxIsFull（pm）
　　integer * 4 pm

说　明：函数 mxIsFull 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxIsFull，用户可以方便地判断由形式参数 pm 指定的阵列是否为满存储类型的阵列，如果是，则函数返回 1，否则返回 0。

29. mxIsNumeric

功　能：判断一个阵列是否为数值类型的阵列。

语　法：integer * 4 function mxIsNumeric（pm）
　　integer * 4 pm

说　明：函数 mxIsNumeric 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxIsNumeric，用户可以方便地判断由形式参数 pm 指定的阵列是否为数值类型的阵列，如果是，则函数返回 1，否则返回 0。

举　例：参见 4.2.2 节的范例程序。

30. mxIsSparse

功　能：判断一个阵列是否为稀疏矩阵。

语　法：integer * 4 function mxIsSparse（pm）
　　integer * 4 pm

说　明：函数 mxIsSparse 为一个 FORTRAN 语言的函数子程序，如果用户希望使用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxIsSparse，用户可以方便地判断由形式参数 pm 指定的阵列是否为稀疏矩阵，如果是，则函数返回 1，否则返回 0。

31. mxIsString

功　能：判断一个阵列是否为字符串类型。

语　法：integer * 4 function mxIsString（pm）
　　integer * 4 pm

说　明：函数 mxIsString，为一个 FORTRAN 语言的函数子程序，如果用户希望使

用它的返回值，必须在程序的开始对该函数进行类型说明。通过函数 mxIsString，用户可以方便地判断由形式参数 pm 指定的阵列是否为字符串类型，如果是，则函数返回 1，否则返回 0。

举 例：参见 4.2.1 节的范例程序。

32. mxSetIr

功 能：设置稀疏矩阵的 ir 数组。

语 法：subroutine mxSetIr (pm, ir)

integer * 4 pm, ir

说 明：函数 mxSetIr 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetIr，用户可以方便地对稀疏矩阵的 ir 数组进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个稀疏矩阵的指针；
- ir 为指向 ir 数组的指针，要求 ir 数组中的元素必须为按列存储。

有关 ir 数组的概念，请读者自行参阅相关章节内容。

33. mxSetJc

功 能：设置稀疏矩阵的 jc 数组。

语 法：subroutine mxSetJc (pm, jc)

integer * 4 pm, jc

说 明：函数 mxSetJc 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetJc，用户可以方便地对稀疏矩阵的 jc 数组进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个稀疏矩阵的指针；
- jc 为指向 jc 数组的指针。

有关 jc 数组的概念，请读者自行参阅相关章节内容。

34. mxSetM

功 能：设置阵列的行数。

语 法：subroutine mxSetM (pm, m)

integer * 4 pm, m

说 明：函数 mxSetM 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetM，用户可以方便地对阵列的行数进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个阵列的指针；
- m 为行数。

这里必须注意的一点是，当对阵列的行数进行扩展之后，必须对 pr、pi、ir

和 jc 这些数组进行相应的扩展；当对阵列的行数减小之后，必须对 pr、pi、ir 和 jc 这些数组进行相应的缩减，以提高内存的利用效率。

举 例：参见 4.2.4 节的范例程序。

35. mxSetN

功 能：设置阵列的列数。

语 法：subroutine mxSetN（pm，n）

integer * 4 pm，n

说 明：函数 mxSetN 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetN，用户可以方便地对阵列的列数进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个阵列的指针；
- n 为列数。

这里必须注意的一点是，当对阵列的列数进行扩展之后，必须对 pr、pi、ir 和 jc 这些数组进行相应的扩展；当对阵列的列数减小之后，必须对 pr、pi、ir 和 jc 这些数组进行相应的缩减，以提高内存的利用效率。

举 例：参见 4.2.4 节的范例程序。

36. mxSetName

功 能：设置阵列的名字。

语 法：subroutine mxSetName（pm，name）

integer * 4 pm

character * （32） name

说 明：函数 mxSetName 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetName，用户可以方便地对阵列的名字进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个阵列的指针；
- name 为一个包含名字的字符串变量。

举 例：参见 mex-函数 mexPutMatrix 的范例程序。

37. mxSetNzmax

功 能：设置稀疏矩阵所能包含的非零元素的最大个数。

语 法：subroutine mxSetNzmax（pm，nzmax）

integer * 4 pm，nzmax

说 明：函数 mxSetNzmax 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetNzmax，用户可以方便地对稀疏矩阵所能包含的非零元素的最大个数进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个稀疏矩阵的指针；
- nzmax 为非零元素的最大个数。

38. mxSetPi

功　能：设置阵列的虚数部分数据。

语　法：subroutine mxSetPi (pm, pi)

integer * 4 pm, pi

说　明：函数 mxSetPi 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetPi，用户可以方便地对阵列的虚数部分数据进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个阵列的指针；
- pi 为指向虚数部分数据的指针。

39. mxSetPr

功　能：设置阵列的实数部分数据。

语　法：subroutine mxSetPr (pm, pr)

integer * 4 pm, pr

说　明：函数 mxSetPr 为一个 FORTRAN 语言的子例行程序，在 MATLAB 接口应用程序中可以直接使用 call 关键字进行调用。通过函数 mxSetPr，用户可以方便地对阵列的实数部分数据进行设置。函数的两个形式参数的含义分别如下：

- pm 为指向一个阵列的指针；
- pr 为指向实数部分数据的指针。

第 5 章　MAT 文件的使用

在本章中，我们将向读者介绍从 MATLAB 中输出数据及输入数据到 MATLAB 中的多种方法，即 MATLAB 与其他应用程序共享数据的各种技巧。这些方法都有实际的应用背景，读者应在充分理解这些方法的使用条件的基础上，应根据实际情况，选择最合适的方法，高效地完成你的工作。而这些技巧中，我们认为是极其重要的，也是 MathWorks 公司极力推荐的方法，就是使用 MAT 文件。

在第 2 章中，我们已经对 MAT 文件进行了简单的介绍。大家可清楚地看到，MAT 文件适用于大部分应用场合。通过对 MAT 文件的使用，我们不仅能够利用 MATLAB 提供的便捷机制，而且 MATLAB 提供了带 mat 前缀的 API 库函数，使我们可以非常容易编写调用这些 API 库函数的 C 和 FORTRAN 语言的应用程序，以便于应用程序与 MATLAB 之间的数据资源共享。

本章将重点介绍如何编写调用这些库函数的 C 或 FORTRAN 语言的应用程序。

5.1　数据的输入和输出

5.1.1　向 MATLAB 输入数据

在使用 MATLAB 时，用户可以采取多种方法将数据传递到 MATLAB 中，以供 MATLAB 使用。如何选择这些方法，应根据数据量的多少和数据已经存储的格式，如是否存为 MATLAB 直接可读的格式等来共同做出决定。下面我们对在输入数据时经常遇到的情况提供以下参考方法：

- 直接在 MATLAB 中使用方括号键入。这种方法非常适合输入少量数据（如不多于 10～15 个元素），并且你非常清楚要输入的数据的值。这种方法对大量数据是不适合的，如果你键入的数据有错，需要及时修改时，却无能为力。
- 第二种方法与第一种方法本质是相同的，就是创建一个用于存放数据的 M 文件。我们可以使用文本编辑器创建一个 M 文件，并在 M 文件中输入数据，而且这种方法允许你使用文本编辑器对数据进行操作，能够及时地修改数据。这种方法特别适合数据还没有以计算机可读方式存在的情况下。
- 如果需要输入的数据已经以 ASCII 码文件形式存在，则可以直接在 MATLAB 中装载文件。MATLAB 提供了直接装载 ASCII 码文件的 load 命令，装载后将在 MATLAB 内存工作区中创建与文件名相同的变量名。在 MATLAB 中，这种文件格式的数据行间距是固定不变的，并采用空格来分离同一行中的数据。我们也可以用文本编辑器来编辑 ASCII 码文件。
- 直接使用 fread、fopen 或 MATLAB 中其他的低级 I/O 函数装载数据。这种方法适合于输入的数据是其他应用程序创建的，特别是该应用程序所特有的数据文件格式。当然，你必须要了解这种数据文件是如何存储的。

- 编写一个 MEX 文件来读这些数据。假如你已经编写了能够从其他的应用程序中读取数据的子程序，这是一个可选用的方法。详细的内容可参看第 2 章。
- 用 C 或 FORTRAN 语言编写程序，用来把你的数据转换变成 MAT 文件格式。我们就能够在 MATLAB 中用 load 命令来装载 MAT 文件中的数据。我们将在本章详细地介绍这种方法的实现。

5.1.2 从 MATLAB 获取数据

同样，外部环境获得 MATLAB 的数据也有多种方法。下面我们对从 MATLAB 中输出数据到外部环境可能会遇到的情况提供以下方法：

- 对于 MATLAB 中需要输出的小矩阵，在 MATLAB 中可以使用 diary 命令，创建 diary 文件。在以后的操作中，生成的 diary 文件可以使用文本编辑器来编辑。需要注意的是，diary 文件的输出包括在会话期间 matlab 的命令行，当然这对于生成文档或报告是有用的。下面是一个 diary 文件的例子：

```
? diary on
    A =
          1    2
          3    4
    ? whos
     Name      Size     Bytes    Class
    A          2×2       32      double array
    Grand total is 4 elements using 32 bytes
    diary ('c: \diary.dat')
```

- 使用 MATLAB 中提供的 save 命令（带 -ascii 选项开关），文件用 ASCII 码的格式来存储数据。我们需要提醒大家注意的是，-ascii 选项仅仅支持数字矩阵形式，对数字 array（超过两维）、cell arrays 和结构都不支持。举例如下：

```
? a=rand (4, 4);
? save test.dat a -ascii
? load test.dat
test =
        0.9501    0.8913    0.8214    0.9218
        0.2311    0.7621    0.4447    0.7382
        0.6068    0.4565    0.6154    0.1763
        0.4860    0.0185    0.7919    0.4057
```

- 直接使用 fread、fwrite 或 MATLAB 中其他低级的 I/O 函数来存储数据文件。这种方法特别适合把数据保存为其他应用程序要求的文件格式。
- 继续利用 MEX 文件来存储数据文件，假如已存在的子程序能够把数据写成其他应用程序可读的文件格式。
- 在 MATLAB 中用 save 命令把数据写成 MATLAB 默认的 MAT 文件。我们可以编写调用 MAT 库函数的 C 或 FORTRAN 语言的应用程序，将 MAT 文件转换成你需要的特殊格式。我们将在本章详细地介绍这种方法的实现。

5.2　MAT 文件应用程序的编写

上一节已经介绍了输入数据到 MATLAB 中和从 MATLAB 中输出数据的多种方法。在这些方法中，使用 MAT 文件作为 MATLAB 与外部环境共享数据的介质，有着无可替代的优势。要使用 MAT 文件，就要利用 MATLAB 所提供的 MAT 库函数进行编程。

MathWorks 公司为帮助用户熟练运用 MATLAB 所提供的 MAT 库函数进行编程，提供了四个典型的范例程序，分别是基于 C 语言的 matcreat.c 和 matdgns.c 以及基于 Fortran 语言的 matdemo1.f 和 matdemo2.f，供大家学习。

本节将首先介绍使用 MAT 库函数需要了解的子目录，然后分别对一个基于 C 语言的例程 matdemo.c 和一个基于 FORTRAN 语言的例程 matdemo.f 进行讲解，来说明如何在 C 语言和 FORTRAN 语言的应用程序中使用 MAT 库函数。

5.2.1　基于 C 语言的 MAT 文件应用程序的编写

1. 程序说明

matdemo.c 是使用 C 语言开发的一个基于 DOS 操作系统的应用程序。这个程序的主要功能是创建一个可被 MATLAB 装载的 MAT 文件，然后读取程序刚创建的 MAT 文件的数据信息。程序由四个程序段组成，第一段是文件的头文件；第二段是创建 MAT 文件的子程序；第三段是读取 MAT 文件信息的子程序；第四段是主函数。

编写这个程序的目的是想向大家介绍在 C 语言中调用 MAT 库函数的方法，并且希望大家尽快掌握以下库函数的使用。

matClose	matGetDir
matGetArray	matGetNextArray
matOpen	matGetNextArrayHeader
matPutArray	matPutArrayAsGlobal

2. 源程序

matdemo.c

```
/* 程序段 (1)　定义程序中用到的头文件 */
#include <stdio.h>
#include "mat.h"
#include "matrix.h"
/* 程序段 (2) */
/* matcreate 子函数，该子函数可以创建能够装载到 MATLAB 中的 MAT 文件 */
int matcreate (const char *file)
{
  /* 代码 1：变量定义及初始化 */
  MATFile *pmatfile;
```

```
mxArray *pa1, *pa2, *pa3;
double data1 [9] = { 1, 2, 3, 4, 5, 6, 7, 8, 9};
double data2 [9] = {11, 12, 13, 14, 15, 16, 17, 18, 19};
printf ("Creating file %s...\n\n", file);
/* 代码 2：创建 MAT 文件 */
pmatfile = matOpen (file, "w");
if (pmatfile == NULL)
{
  printf ("Error creating file %s\n", file);
  return (1);
}
/* 代码 3：创建 mxArray 类型的数组，并给数组指针赋值及对数组命名 */
pa1 = mxCreateDoubleMatrix (3, 3, mxREAL);
mxSetName (pa1, "LDdata");
memcpy ( (char * ) (mxGetPr (pa1)), (char * ) data1, 3 * 3 * sizeof (double));
pa2 = mxCreateDoubleMatrix (3, 3, mxREAL);
mxSetName (pa2, "GDdata");
memcpy ( (char * ) (mxGetPr (pa2)), (char * ) data2, 3 * 3 * sizeof (double));
pa3 = mxCreateString ("We teach you use the mat function.");
mxSetName (pa3, "GSdata");
/* 代码 4：把代码 3 得到的 mxArray 数组写入到 MAT 文件中 */
matPutArray (pmatfile, pa1);
matPutArrayAsGlobal (pmatfile, pa2);
matPutArrayAsGlobal (pmatfile, pa3);
/* 代码 5：删除已经赋值的指针变量 */
mxDestroyArray (pa1);
mxDestroyArray (pa2);
mxDestroyArray (pa3);
/* 代码 6：关闭在代码 2 中创建的 MAT 文件 */
  if (matClose (pmatfile) ! = 0)
  {
    printf ("Error closing file %s\n", file);
    return (1);
    }
  }
/* 程序段 (3) */
/* 子函数 matread：读取已经存在的 MAT 文件中的信息 */
int matread (const char *file)
 {
   /* 代码 1：变量定义 */
   MATFile *pmat;
   char **dir;
   int ndir;
   int i;
```

```
mxArray *pa;
printf ("Reading file %s....\n\n", file);
/* 代码2：打开MAT文件 */
pmat=matOpen (file,"r");
if (pmat==NULL)
{
   printf ("Error opening file %s\n", file);
   return (1);
}
/* 代码3：从打开的MAT文件中，获得所存储的所有数组名 */
dir=matGetDir (pmat, &ndir);
if (dir==NULL)
{
   printf ("Error reading directory of file %s\n", file);
   return (1);
   }
else
{
   printf ("Directory of %s\n", file);
   for (i=0; i<ndir; i++)
       printf ("%s \n", dir [i]);
}
mxFree (dir);
if (matClose (pmat)! =0)
{
   printf ("Error closing file %s\n", file);
   return (1);
}
/* 代码4：从文件头中读出存储的数组信息 */
printf ("\nExamining the header for each variable: \n");
pmat=matOpen (file,"r");
for (i=0; i<ndir; i++)
{
   pa=matGetNextArrayHeader (pmat);
   if (pa==NULL)
   {
    printf ("Error reading in file %s\n", file);
    return (1);
   }
 printf ("According to its header , array %s has %d dimensions\n",
    mxGetName (pa),
    mxGetNumberOfDimensions (pa));
 if (mxIsFromGlobalWS (pa))
    printf (" and was a global variable when saved\n");
```

```
    else
      printf (" and was a local variable when saved\n");
    mxDestroyArray (pa);
    }
    if (matClose (pmat)! =0)
    {
      printf ("Error closing file %s \n", file);
      return (1);
    }
    /* 代码 5：直接读出文件中的数组信息 */
    pmat=matOpen (file,"r");
    if (pmat==NULL)
    {
      printf ("Error reopening file %s\n", file);
      return (1);
    }
    printf ("\nReading in the actual array contents: \n");
    for (i=0; i<ndir; i++)
    {
      pa=matGetNextArray (pmat);
      if (pa==NULL)
      {
        printf ("Error reading in file %s\n", file);
        return (1);
      }
      printf ("According to its contents,
        arrray %s has %d dimensions\n", mxGetName (pa),
        mxGetNumberOfDimensions (pa));
      if (mxIsFromGlobalWS (pa))
        printf (" and was a global variable when saved\n");
      else
        printf (" and was a local variable when saved\n");
      mxDestroyArray (pa);
    }
    if (matClose (pmat)! =0)
    {
      printf ("Error closing file %s \n", file);
      return (1);
    }
    return (0);
}
/* 程序段 (4) */
int main ()
 {
```

```
    int createresult;
    int readresult;
    int result;
    /* 分别调用创建MAT文件的子函数和读取MAT文件的子函数 */
    createresult = matcreate ("mattest.mat");
      readresult = matread ("mattest.mat");
      if ( (createresult==0) && (readresult==0))
           result=0;
    else
           result=1;
    return (result==0)? EXIT_SUCCESS: EXIT_FAILURE;
  }
```

3. *源程序分析*

下面将分别对四个程序段的功能进行分析及对源程序中使用到的 MAT 库函数的用法进行说明。

· 程序段（1）：文件包含处理代码

这段代码对程序中所必需的一些头文件进行了包含，其中最为重要的是 matrix.h 和 mat.h 两个头文件。matrix.h 中包含了程序中使用到的操作 MATLAB 数组的库函数的函数原型定义。mat.h 中包含了程序中使用到的操作 MAT 文件的库函数的函数原型定义。缺了这两个头文件，程序将无法编译。

· 程序段（2）：子函数 matcreate

子函数的作用为创建 MAT 文件，并将变量名和变量值写入到创建的 MAT 文件中。子函数的函数名为 matcreate；形式参数 file 为指向常字符类型指针变量。当主调函数调用该子函数时，将要创建的文件名传递给 file。

该子函数由以下代码段组成，下面将分别介绍各代码段的功能，并对各代码段中用到的 mat 库函数的使用进行说明。

代码 1 是变量定义及初始化模块，其中 pmatfile 为指向 MATFile 的指针变量；pa1、pa2 和 pa3 为指向 mxArray 结构体类型数据的指针变量；data1 和 data2 为 double 数据类型的变量。

代码 2 是创建 MAT 文件模块，同时检验 MAT 文件的创建情况。和其他高级语言一样，对文件读写之前应该“打开”文件，在使用结束之后应关闭该文件。打开文件是利用 MATLAB 提供的库函数 matOpen。它表示：要打开名字为 file 的文件，使用文件的方式为“只写”，matOpen 库函数带回指向 file 文件的指针并赋给 pmatfile。注意，如果要打开的文件并不存在，matOpen 库函数将创建一个以参数 file 为文件名的 MAT 文件。如果不能实现“打开”任务，matOpen 库函数将会带回一个出错信息：一个空指针 NULL。这里先检查打开是否出错，如果有错就在终端上输出“Error creating file”，并返回主调函数。

代码 3 是创建变量和变量命名模块。该模块中，分别用库函数 mxCreateDoubleMatrix 和 mxCreateString 创建 Double 类型的变量和 String 类型的变量，并分别将创

建的变量指针赋给变量pa1、pa2和pa3。同时使用库函数mxSetName对变量命名：pa1所指的变量名为“LDdata”；pa2所指的变量名“GDdata”；pa3所指的变量名为“GSdata”。并分别对变量赋值。

代码4为将数组名、数组值写入到MAT文件中的功能模块。其中pa1所指变量用matPutArray库函数写入，pa2和pa3所指的变量用matPutArrayAsGlobal库函数写入。matPutArray库函数允许用户将一个mxArray数组写入到MAT文件中，当MAT文件被MATLAB装载时，这个被写入的变量为局部变量；matPutArrayAsGlobal函数与matPutArray函数的使用方法相似，惟一不同的是，MATLAB装载这个数组时，将该变量装载到全局内存工作区。

代码5为删除已经赋值的指针变量的功能模块，在该模块中，分别删除了pa1、pa2和pa3，释放占用的系统资源。

代码6是关闭MAT文件模块，它对文件关闭成功与否进行判断。matClose库函数关闭一个打开的MAT文件，该函数返回一个整型值。当文件成功关闭时，其值为0，否则其值为非0值。

· 程序段（3）：子函数matread

该子函数的作用为读取MAT文件的内容。该子函数的形式参数file为指向常字符类型的指针变量。当子函数被调用时，主调函数把mat文件名传递给file。该子函数返回值类型为整型，当函数返回值为1时，该子函数调用失败；当函数返回值为0时，该子函数成功返回。

代码1是变量定义及初始化模块。同子函数matcreat一样，pmat变量为MATFile的指针变量；pa为指向mxArray数据类型的指针变量；dir变量定义为指向字符指针的指针变量；i和ndir定义为整型数据类型的变量。

代码2的功能与子函数matcreat代码2相似，惟一不同的地方是文件的使用方式不同，这里为“只读”。

代码3是获取变量名模块。matGetDir库函数得到被打开的MAT文件里存储的所有mxArray结构体类型变量的变量名，并且返回指向字符串指针数组，该数组中存放了所有变量的变量名，并将它赋给dir。MAT文件中变量的个数将被存放到第二个参数ndir指向的变量中。此外该子函数中使用了函数mxCalloc来分配返回指针地址，程序结束时，必须使用mxFree来释放占用的内存资源。

代码4为从数组头中读出存储的变量信息的功能模块。matGetNextArrayHeader库函数从MAT文件中按存储的先后顺序读取变量头中的变量信息，信息中包括除了pr，pi，ir及jc的所有信息，同时该库函数将变量的指针返回并赋给pa。当函数调用失败时，将返回NULL。然后，逐个对读出的数组头信息进行判断，包括维数、变量类型，以便与代码5直接读出文件中的数组信息进行校验。

代码5直接读出文件中的变量信息。matGetNextArray库函数从MAT文件中按存储的先后顺序直接读取变量的变量信息，并将该变量的指针返回赋给pa。使用中需要注意的是该函数应该在matOpen库函数使用后，立即使用，不能在其他的MAT库函数使用后再使用，并且该函数是使用mxCalloc来分配指针地址，程序中必须释放占用的内存资源。用户可对该代码段中读取的信息与代码4进行比较，看两次的内容是否一致。

• 程序段（4）：主函数

主调函数首先调用创建 MAT 文件的子函数 matcreat，并将文件名 mattest.mat 传递给形参。然后再调用读取 mattest.mat 文件信息的子函数 matread，读取创建的 mattest.mat 文件的信息。其中 result 的值将帮助我们判断调用的子函数是否成功返回。

4. *程序执行结果*

程序经编译和链接后，形成基于 DOS 的可执行程序。运行该可执行程序后，可在 MATLAB 中用 whos 命令，查看程序形成的 MAT 文件。查看结果如下：

```
? whos -file mattest.mat
   Name        Size         Bytes Class
   GDdata      3x3             72 double array (global)
   GSdata      1x34            68 char array (global)
   LDdata      3x3             72 double array
```

5.2.2　基于 FORTRAN 语言的 MAT 库函数的使用例程

1. *程序说明*

matdemo.f 是使用 Fortran 语言开发的一个基于 DOS 操作系统的应用程序。这个程序的主要功能是创建一个可被 MATLAB 装载的 MAT 文件，并读取 MAT 文件中的信息。

编写这个程序的目的是想向大家介绍在 FORTRAN 语言中调用 MAT 库函数的方法，并且希望大家尽快掌握以下库函数的使用。

matOpen	matClose
matGetDir	matGetNextMatrix
matPutMatrix	

2. matdemo.f *源程序*

```
C       matdemo.f 程序将创建一个可被 matlab 装载的 mat 文件，
C       并读取新创建的 mat 文件中的内容
C
        program matdemo
C
C       程序段（1）：定义外部函数和变量的类型及对部分变量赋值。
        integer matOpen, matClose
        integer mxCreateFull
        integer matGetMatrix, mxGetPr
C
        integer matGetDir, matGetNextMatrix, mxCreateString
        integer mxGetM, mxGetN
C
         integer stat
        integer mp, pa1, pa2, pa3, pa4
```

```
      integer ndir, i
      integer dir, adir (100)
      character * 32 names (10), name, mxGetName
C
      double precision dat1 (9), dat2 (4)
      data dat1/1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0/
      data dat2/10.0, 11.0, 12.0, 13.0/
C
C     程序段 (2): 创建一个新的 MAT 文件 matdemo.mat。
      write (6, * ) 'Now creating MAT-file matdemo.mat...'
      mp=matOpen ('matdemo.mat','w')
      if (mp.eq.0) then
         write (6, * ) 'Can't open matdemo.mat for writing'
         write (6, * ) ' (Do you have write permission in this directory?)'
         stop
      end if
C
C     程序段 (3): 创建 mxArray 结构体类型的对象,并给该对象赋值和命名
      pa1=mxCreateFull (3, 3, 0)
      call mxCopyReal8ToPtr (dat1, mxGetPr (pa1), 9)
      call mxSetName (pa1,'data1')
      pa2=mxCreateFull (2, 2, 0)
      call mxCopyReal8ToPtr (dat2, mxGetPr (pa2), 4)
      call mxSetName (pa2,'data2')
      pa3=mxCreateString ('This is a sample how to use MAT-file.')
      call mxSetName (pa3,'string')
C
C     程序段 (4): 将新创建的 mxArray 结构体对象写入到 MAT 文件
C                 matdemo.mat 文件中
      call matPutMatrix (mp, pa1)
      call matPutMatrix (mp, pa2)
      call matPutMatrix (mp, pa3)
C
C     程序段 (5): 关闭 MAT 文件 matdemo.mat
      stat=matClose (mp)
      if (stat.ne.0) then
         write (6, * )'Error closing MAT-file'
         stop
      end if
C
C     程序段 (6): 重新打开 matdemo.mat 文件,并从该文件中得到 mxArray
C                 结构体对象列表
      mp=matOpen ('matdemo.mat','r')
      if (mp.eq.0) then
         write (6, * )'Can't open matdemo.mat for reading.'
```

```
            stop
          end if
C
          dir=matGetDir (mp, ndir)
          if (dir.eq.0) then
            write (6, *)'Can't read directory.'
            stop
          end if
C
          call mxCopyPtrToPtrArray (dir, adir, ndir)
C
          do 10 i=1, ndir
          call mxCopyPtrToCharacter (adir (i), names (i), 32)
   10     continue
C
          write (6, *)'Directory of MAT-file:'
          do 20 i=1, ndir
              write (6, *) names (i)
   20   continue
C
          stat=matClose (mp)
          if (stat.ne.0) then
            write (6, *)'Error closing matdemo.mat.'
            stop
          end if
C
C         程序段 (7)：再一次打开 matdemo.mat 文件，直接读取该文件的信息
          mp=matOpen ('matdemo.mat','r')
          if (mp.eq.0) then
            write (6, *)'Can't open matdemo.mat.'
            stop
          end if
C
          write (6, *)'Getting full array contents:'
          pa4=matGetNextMatrix (mp)
C
          do while (pa4.ne.0)
            name=mxGetName (pa4)
            write (6, *)'Retrieved', name
            write (6, *)'With Size', mxGetM (pa4),'-by-', mxGetN (pa4)
            pa4=matGetNextMatrix (mp)
          end do
C
          stat=matClose (mp)
          if (stat.ne.0) then
```

```
            write (6, * )'Error closing "matdemo.mat".'
            stop
        end if
        stop
C
        end
```

3. *源程序分析*

该程序由七个程序段组成，下面将分别介绍各程序段的功能，并对各程序段中用到的MAT库函数的使用进行说明。

程序段(1)是外部函数和变量定义模块，其中包括部分变量的初始化。在FORTRAN程序中，函数子程序名既是某个函数的名字又代表该函数的函数值，所以在需要使用该函数的返回值时，必须对其进行类型说明。如果省略了对函数名的类型说明，则按隐含类型规则由函数名的第一个字母来确定函数值的类型。程序中用call命令调用的子函数属于子例行程序，子例行程序的名字只供调用，它不代表某个值，当然也不属于某个类型，因此在程序中不需要定义类型。该代码段首先定义了程序中要用到的MAT库函数及mx-库函数的函数类型，接着定义了程序中将要使用到的变量类型，并对部分变量进行了赋初值。

程序段(2)是创建一个新的MAT文件模块，同时检验MAT文件的创建情况。和其他高级语言一样，对文件读写之前应该“打开”该文件，在使用结束之后应关闭该文件。打开文件是利用MATLAB提供的库函数matOpen。它表示：要打开名字为matdemo.mat的文件，使用文件的方式为“只写”。注意，如果要打开的文件并不存在，matOpen库函数将创建一个以参数file为文件名的MAT文件。如果不能实现“打开”任务，matOpen库函数将会带回一个出错信息：一个整型值0。这里先检查打开是否出错，如果有错就在终端上输出“Error creating file”。

程序段(3)创建mxArray结构体类型的对象，并给该对象赋值和命名。通过使用库函数mxCreateFull和mxCreateString创建Double类型的变量和String类型的mxArray结构体对象，并分别将前面创建的数组赋给变量pa1、pa2和pa3。接下来分别对变量赋值。同时通过使用库函数mxSetName对创建的数组命名，pa1命名为“data1”；pa2命名“data2”；pa3命名为“string”。

程序段(4)是将新创建的mxArray结构体对象写入到MAT文件中的功能模块。pa1、pa2和pa3的写入工作通过库函数matPutMatrix完成，matPutMatrix能够把一个mxArray型变量写入到MAT文件中。假如在MAT文件中不存在与将要写入的数组具有相同数组名的数组，则这个将要存储的数组被放置在文件的末尾；如果MAT文件存在同名的数组，这个新写入的数组将替换旧数组，不管它们的尺寸大小是否一致。

程序段(5)是关闭MAT文件的功能模块，并对文件关闭成功与否进行判断。matClose库函数关闭一个打开的MAT文件。该函数返回一个整型值，当文件成功关闭时其值为0，否则其值为非0值。

程序段(6)重新打开matdemo.mat文件，并从文件中得到mxArray结构体对象名的列表。这里matOpen库函数的使用与代码(2)中相似，不同的地方是文件的使用方式

不同，这里是以“只读”方式打开文件。matGetDir 库函数得到被打开的 MAT 文件里存储的所有 mxArray 结构体对象名的列表。该库函数返回指向字符串数组（存储了变量名的数组）的指针，并将它赋给 dir。该 MAT 文件中变量的个数将被存放到第二个参数 ndir 指向的变量中。

程序段(7)再一次打开 matdemo.mat 文件，直接读取该文件的信息。matGetNextMatrix 库函数从 MAT 文件中按存储的先后顺序直接读取变量的变量信息，并将该变量的指针返回赋给 pa4。使用中需要注意的是该函数应该在 matOpen 库函数使用后立即使用，不能在其他的 MAT 库函数使用后再使用，否则下一个数组的概念将没有意义。

4. 程序执行结果

程序经编译和链接后，形成基于 DOS 的可执行程序。运行该可执行程序后，可在 MATLAB 中用 whos 命令，查看程序形成的 MAT 文件。查看结果如下：

```
? whos
   Name      Size           Bytes Class
   data1     3x3               72 double array
   data2     2x2               32 double array
   string    1x37              74 char array
  Grand total is 50 elements using 178 bytes
```

5.3 MAT 文件应用程序的建立和调试

在本节中，我们将分别基于 C 语言和 FOTRAN 语言，对 MATLAB 引擎程序的建立和调试方法进行介绍，并给出直接在 Microsoft VC++ 6.0 集成环境和 Microsoft Fortran PowerStation 集成环境中建立和调试 MATLAB 引擎程序的方法。

5.3.1 C 语言 MAT 文件应用程序的建立和调试

1. C 语言 MAT 文件应用程序的建立

在 Windows 操作系统上，C 语言 MAT 文件应用程序的建立极为简单，用户在编写好源程序后，只需在 MATLAB 命令提示符下键入命令 mex 并辅以参数-f 和相应的选项文件以及需要编译的源程序名即可，格式如下：

mex -f <matlab>\bin*optsfilename*.bat *filename*.c

其中 *optsfilename*.bat 为与用户系统中 C 语言编译器相对应的选项文件名，*filename*.c 为用户编写的 MAT 文件应用程序的源文件名，参数-f 的含义为使用指定的选项文件对 *filename*.c 进行编译。关于选项文件的选择，读者可以参见本书的第 2.4.1 节。若应用程序编译成功，mex 命令将不返回任何信息，直接回到 MATLAB 命令提示符下，否则将给出一定的错误原因。

2. C 语言 MAT 文件应用程序的调试

C 语言 MAT 文件应用程序在编译链接通过后，并不意味着已经完全没有错误，这一

点对于有经验的程序员来说是显而易见的。如果程序在运行时发生错误，最简单直接的查错方法就是调试（DEBUG），在一般的应用程序编制过程中，这是一个必不可少的步骤。下面我们以Microsoft VC++ 6.0为例，对C语言MAT文件应用程序的调试进行讲解。

从大体上整个过程可以分为两步：首先开启集成编译环境，选择需要调试的MAT文件应用程序的执行文件，将其调入到集成环境中，这时VC将为其创建一个工作空间；其次选择MAT文件应用程序的源文件，将其调入到当前工程中。至此就完成了全部的设置工作，接下来就可以在程序中设置断点进行调试了。关于调试器的使用，请读者自行参见相关调试器的联机帮助。

这里必须非常注意一点，如果希望对一个C语言MAT文件应用程序进行调试，那么在使用mex命令进行编译时，必须同时加入命令参数-g，格式如下：

mex -g -f <matlab>\bin*optsfilename*.bat *filename*.c

其中参数-g的含义为告诉编译器在编译链接程序时包含入调试信息，否则程序将无法调试。强烈建议用户在首次调试程序时使用-g参数。

3. 对选项文件msvc60engmatopts.bat的分析

选项文件msvc60engmatopts.bat是MATLAB为Microsoft Visual C++ 6.0提供的一个专门用于编译MAT文件的选项文件，位于目录

<MATLAB根目录>\BIN

中，其源代码如下，请读者先详细阅读该程序，在后面将对该选项文件进行全面的分析，以加深读者对C语言MAT文件应用程序编译过程的理解。

```
@echo off
rem MSVC60ENGMATOPTS.BAT
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rem (1) 普通参数设置
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
set MATLAB=%MATLAB%
set MSVCDir=%MSVCDir%
set MSDevDir=%MSVCDir%\..\Common\msdev98
set PATH=%MSVCDir%\BIN;%MSDevDir%\bin;%PATH%
set INCLUDE =%MSVCDir%\INCLUDE;%MSVCDir%\MFC\INCLUDE;
             %MSVCDir%\ATL\INCLUDE;%INCLUDE%
set LIB=%MSVCDir%\LIB;%MSVCDir%\MFC\LIB;%LIB%

rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rem (2) 编译器参数设置
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

set COMPILER = cl
set OPTIMFLAGS = -O2
set DEBUGFLAGS = -Zi
set COMPFLAGS = -c -Zp8 -G5 -W3 -nologo
set NAME_OBJECT = /Fo
```

```
rem ******************************************
rem (3) 库创建命令 (预编译过程)
rem ******************************************

set PRELINK _CMDS =lib /def:"%MATLAB%\extern\include\libmx.def" /machine: ix86
                    /OUT:%LIB _NAME%1.lib /NOLOGO
set PRELINK _CMDS =%PRELINK _CMDS%; lib /def:"%MATLAB%\extern\include
                    \libeng.def" /machine: ix86 /OUT:%LIB _NAME%2.lib /NOLOGO
set PRELINK _CMDS =%PRELINK _CMDS%; lib /def:"%MATLAB%\extern\include
                    \libmat.def" /machine: ix86 /OUT:%LIB _NAME%3.lib /NOLOGO
set PRELINK _DLLS=lib /def:"%MATLAB%\extern\include\%DLL _NAME%.def"
                    /machine: ix86 /OUT:%DLL _NAME%.lib /NOLOGO

rem ******************************************
rem (4) 链接器参数设置
rem ******************************************

set LINKER = link
set LINKFLAGS = kernel32.lib user32.lib gdi32.lib %LIB _NAME%1.lib
        %LIB _NAME%2.lib %LIB _NAME%3.lib /implib:%LIB _NAME%l.lib /nologo
set LINKOPTIMFLAGS =
set LINKDEBUGFLAGS = /debug
set LINK _FILE =
set LINK _LIB =
set NAME _OUTPUT = "/out:%OUTDIR%%MEX _NAME%.exe"
set RSP _FILE _INDICATOR = @

rem ******************************************
rem (5) 资源编译器参数设置
rem ******************************************

set RC _COMPILER = rc        /fo "%OUTDIR%mexversion.res"
set RC _LINKER =
```

由上面的源代码可以看出，选项文件 msvc60engmatopts.bat 的结构非常清晰，总共可以分为五个部分：

第一部分为普通参数设置，定义了一些关于 MATLAB 以及 C 语言编译器的路径信息，其中%MATLAB%和%MSVCDir%分别代表了 MATLAB 和 Microsoft Visual C++ 6.0 所在的安装路径，其余的一些相关定义，都是建立在它们的基础之上；

第二部分为编译器参数设置，在该选项文件中，通过环境变量 COMPILER 设置了编译器为 cl，并且通过环境变量 COMPFLAGS、OPTIMFLAGS、和 DEBUGFLAGS 对编译器进行了全面的设置，其中各参数的含义分别如下：

-G5　代表针对奔腾处理器进行优化处理

-W3　代表设置警告级别为 3 级

-Zi　代表使能调试信息

-O2　代表以最大速度优化

-nologo　代表禁止版权信息

-c 代表告诉编译器，只对源文件进行编译而不用链接
-Zp8 代表按8个字节对准

第三部分为库创建命令，即预编译过程，用于创建输入函数库；

第四部分为链接参数设置，在这部分内容中，首先通过环境变量LINKER设置了链接器的名字为link，然后通过环境变量LINKFLAGS设置了链接器的一些参数选项，声明了一些在生成MAT文件的应用程序时所需要嵌入的库文件，包括在第三部分中生成的库文件；

第五部分为资源编译器参数设置。

4. Microsoft VC++ 6.0 集成环境中MAT文件应用程序的建立和调试

习惯了使用各种集成环境的读者，可能会觉得MATLAB提供的这种程序开发方法非常的不方便。建立一个MAT文件应用程序，首先必须在某个编辑器中进行源代码的编写，然后存盘后回到MATLAB的工作环境中，进行编译，若发现错误，则必须回到原来的文件编辑器，按照MATLAB的错误提示，逐行地对源代码进行查错修改，之后再回到MATLAB的工作环境中进行编译，如此往复，直至程序没有错误为止，一个MAT文件应用程序才宣告完成。整个过程需要来回地在文件编辑器和MATLAB工作环境之间切换，非常不方便，而且过程中还没有加入调试步骤，否则将更加麻烦。

为了方便读者，我们将给出一种在Microsoft VC++ 6.0集成环境中的建立和调试MAT文件应用程序的方法，其基本步骤如下：

第一步，启动Microsoft VC++ 6.0集成环境，选择File下拉式菜单中的New选项，这时将弹出如图5.1所示的对话框，用户可以选择其中三种类型的应用程序创建工程，分别为MFC AppWizard (exe)，Win32 Application和Win32 console Application，为了简便起见，我们以Win32 console Application为例进行说明。选择Win32 console

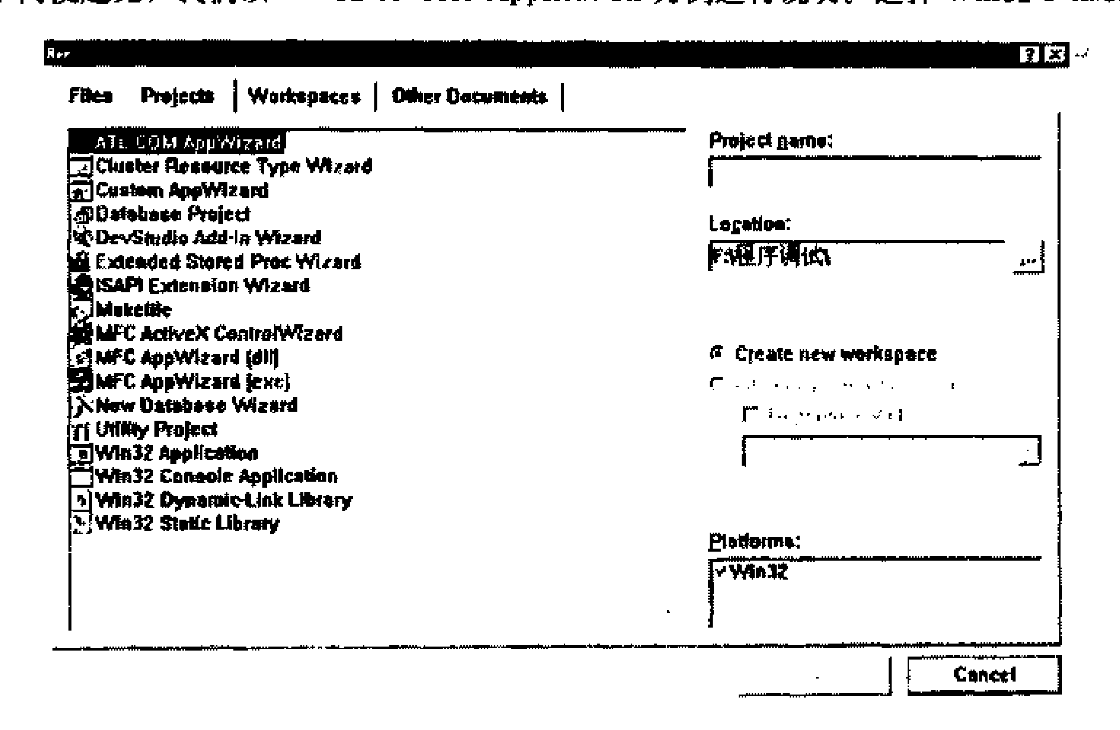


图5.1 VC++ 6.0 File New 对话框

Application 项目，并在 Project name 编辑框中输入项目名，并点击 OK 按钮；

第二步，完成以上操作后，VC++ 6.0 编译器将弹出如图 5.2 所示的对话框，提示用户选择创建 Win32 console Application 程序的类型，这里我们选择其中的最为简单的类型 An Empty project，然后选择 Finish 按钮，创建该项目。

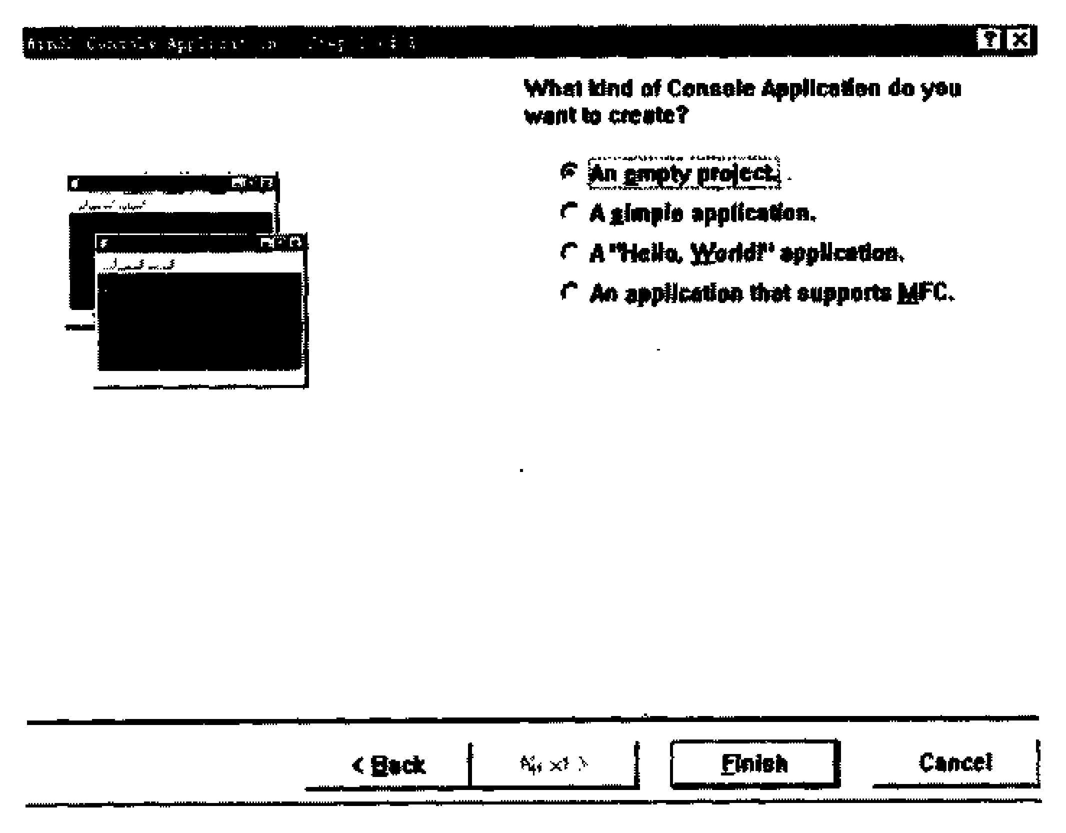


图 5.2　Win32 console Application 程序的类型选择对话框

第三步，在项目工程创建完毕之后，选择下拉式菜单 Tools 中的菜单项 Options，将弹出 Options 对话框，选择其中的 Directories 属性页，如图 5.3，在其中的 Show directories for 下拉式选项框中分别选择 Include Files 和 Library Files，在下部的编辑框中输入以下路径：

MATLAB 根目录\EXTERN\INCLUDE

MATLAB 根目录\EXTERN\LIB

然后选择 OK 按钮；

第四步，在 DOS 命令框状态下，进入用户安装 Microsoft VC++ 6.0 的目录，如 d：\Program files\Microsoft Visual Studio\VC98，并且进入该目录下的子目录\bin，按下面的格式运行该目录下的命令 lib：

lib /def：%MATLAB%\extern\include\libmx.def /machine：ix86 /OUT：LIB_NAME1.lib /NOLOGO

lib /def：%MATLAB%\extern\include\libeng.def /machine：ix86 /OUT：LIB_NAME2.lib /NOLOGO

lib /def：%MATLAB%\extern\include\libmat.def /machine：ix86 /OUT：LIB_NAME3.lib /NOLOGO

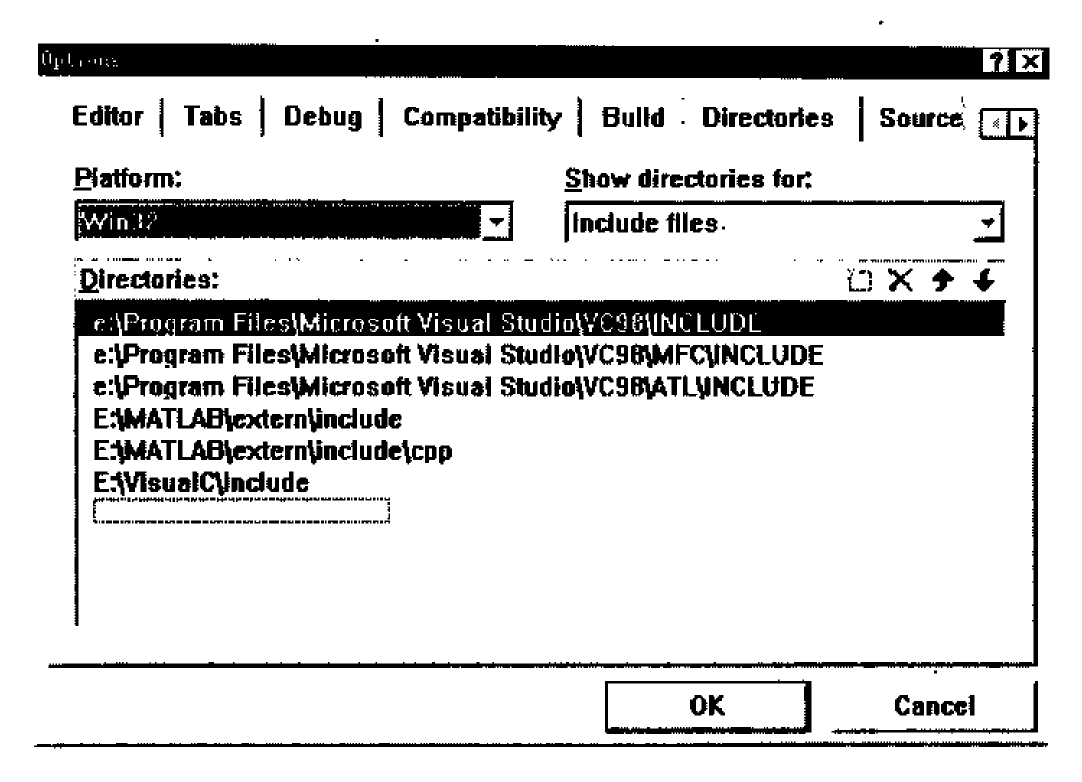


图5.3 Options Directories 属性页

执行完这三条命令后用户可以得到三个静态链接库文件，分别为LIB _NAME1.lib、LIB _NAME2.lib 和LIB _NAME3.lib。命令中%MATLAB%代表本机上安装MATLAB的根目录，在执行这些命令的过程中，必须加以替换，如D：\MATLAB；

这里可以明确一点，一旦三个静态链接库文件生成后，就可以反复使用，而无须对每一个项目进行重新建立；

第五步，选择下拉式菜单Project 中的菜单项Add To Project>>Files，将第四步中生成的三个lib 库文件添加到当前项目中，同时将用户编写的MAT 文件应用程序的源文件也添加到当前项目中。

当完成了以上五步工作之后，用户就可以在VC++中对MAT 文件应用程序进行编译和调试了。对于MFC Application 和Win32 Application 类型的MAT 文件应用程序，步骤大致相同。

5.3.2 FORTRAN 语言MAT文件应用程序的建立和调试

1. FORTRAN 语言MAT 文件应用程序的建立

在Windows 操作系统上，FORTRAN 语言MAT 文件应用程序的建立与C 语言MAT 文件应用程序的建立过程极为相似。用户在编写好源程序后，只需在MATLAB命令提示符下键入命令mex 并辅以参数-f 和相应的选项文件以及需要编译的源程序名即可，格式如下：

mex -f <matlab>\bin*optsfilename*.bat *filename*.f

惟一的不同就是命令中*optsfilename*.bat 为与用户系统中FORTRAN 语言编译器相对应的选项文件名，*filename*.f 为用户编写的MAT 文件应用程序的源文件名，参数-f 的含义为使用指定的选项文件对*filename*.f 进行编译。关于选项文件的选择，读者可以参见

本书的第 2.4.1 节。若应用程序编译成功，mex 命令将不返回任何信息，直接回到 MATLAB 命令提示符下，否则将给出一定的错误原因。

2. FORTRAN 语言 MAT 文件应用程序的调试

与 C 语言 MAT 文件应用程序类似，FORTRAN 语言 MAT 文件应用程序在编译链接通过后，并不意味着已经完全没有错误，同样需要一个调试过程。下面我们以 Microsoft Fortran PowerStation 为例，对 FORTRAN 语言 MAT 文件应用程序的调试进行说明。

整个调试过程具体可以分为两步：首先进入 Microsoft Fortran PowerStation 集成环境，选择下拉式菜单 File 中的菜单项 Open Workspace，这时将弹出一个文件选择对话框，用户从中选择希望调试的 MAT 文件应用程序的可执行文件，将其调入当前工作空间；第二步，选择下拉式菜单 File 中的菜单项 Open 将 MAT 文件应用程序的源程序调入工作空间。至此就完成了全部的设置工作，接下来就可以在程序中设置断点进行调试了。关于调试器的使用，请读者自行参见相关调试器的联机帮助。

这里同样必须非常注意一点，如果希望对一个 FORTRAN 语言 MAT 文件应用程序进行调试，那么在使用 mex 命令进行编译时，必须同时加入命令参数-g，格式如下：

mex -g -f <matlab>\bin*optsfilename*.bat *filename*.c

其中参数-g 的含义为告诉编译器在编译链接程序时包含人调试信息，否则程序将无法调试。强烈建议用户在首次调试程序时使用-g 参数。

3. 对选项文件 df50engmatopts.bat 的分析

df50engmatopts.bat 是 MATLAB 为 DEC Fortran 5.0 编译器提供的专门编译 FORTRAN 语言 MAT 文件应用程序的选项文件，位于目录

<MATLAB 根目录>\BIN

中，其源代码如下，请读者先详细阅读该程序，在后面将对该选项文件进行全面的分析，以加深读者对 FORTRAN 语言 MAT 文件应用程序编译过程的理解。

```
@echo off
rem DF50ENGMATOPTS.BAT

rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rem (1) 普通参数设置
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

set MATLAB=%MATLAB%
set DF_ROOT=%DF_ROOT%
set PATH = %DF_ROOT%\sharedIDE\bin;%DF_ROOT%\DF\BIN;
                    %DF_ROOT%\VC\BIN;%PATH%
set INCLUDE=%DF_ROOT%\DF\INCLUDE;%INCLUDE%
set LIB=%DF_ROOT%\DF\LIB;%DF_ROOT%\VC\LIB;%LIB%

rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rem (2) 编译器参数设置
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

set COMPILER=fl32
```

```
set OPTIMFLAGS=-Oxp
set DEBUGFLAGS=-Zi
set COMPFLAGS=-c -G5 -4R8 -nologo
set NAME _OBJECT=/Fo

rem ****************************************
rem (3) 库创建命令 (预编译过程)
rem ****************************************

set PRELINK _CMDS=lib /def:%MATLAB%\extern\include\dfmx.def
                    /machine: ix86 /OUT:%LIB _NAME%1.lib /NOLOGO
set PRELINK _CMDS=%PRELINK _CMDS%; lib /def:%MATLAB%\extern\include
                    \dfmat.def /machine: ix86 /OUT:%LIB _NAME%2.lib /NOLOGO
set PRELINK _CMDS=%PRELINK _CMDS%; lib /def:%MATLAB%\extern\include
                    \dfeng.def /machine: ix86 /OUT:%LIB _NAME%3.lib /NOLOGO
set PRELINK _DLLS=lib /def:%MATLAB%\extern\include\%DLL _NAME%.def
                    /machine: ix86 /OUT:%DLL _NAME%.lib /NOLOGO

rem ****************************************
rem (4) 链接器参数设置
rem ****************************************

set LINKER=link
set LINKFLAGS= %LIB _NAME%1.lib %LIB _NAME%2.lib %LIB _NAME%3.lib
set LINKOPTIMFLAGS=
set LINKDEBUGFLAGS=/debug
set LINK _FILE=
set LINK _LIB=
set NAME _OUTPUT="/out:%OUTDIR%%MEX _NAME%.exe"
set RSP _FILE _INDICATOR=@

rem ****************************************
rem (5) 资源编译器参数设置
rem ****************************************

set RC _COMPILER=rc /fo "%OUTDIR%mexversion.res"
set RC _LINKER=
```

由上面的源代码可以看出，选项文件 df50engmatopts 的结构非常清晰，总共可以分为五个部分：

第一部分为普通参数设置，定义了一些关于 MATLAB 以及 FORTRAN 语言编译器的路径信息，其中%MATLAB%和%DF _ROOT%分别代表了 MATLAB 和 DEC Fortran 5.0 编译器所在的安装路径，其余的一些相关定义，都是建立在它们的基础之上；

第二部分为编译器参数设置，在该选项文件中，通过环境变量 COMPILER 设置了编译器为 fl32，并且通过环境变量 COMPFLAGS、OPTIMFLAGS、和 DEBUGFLAGS 对编译器进行了全面的设置，其中各参数的含义分别如下：

-G5　代表针对奔腾处理器进行优化处理

-Zi　代表使能调试信息

-Oxp　代表对源代码进行全面优化

-nologo　代表禁止版权信息

-c　代表告诉编译器，只对源文件进行编译而不用链接

第三部分为库创建命令，即预编译过程，用于创建输入函数库；

第四部分为链接参数设置，在这部分内容中，首先通过环境变量 LINKER 设置了链接器的名字，为 link，然后通过环境变量 LINKFLAGS 设置了链接器的一些参数选项，声明了一些在生成 MAT 文件的应用程序时所需要嵌入的库文件，包括在第三部分中生成的库文件；设置环境变量 LINKDEBUGFLAGS 为/debug，表示包含调试信息；

第五部分为资源编译器参数设置。

4. Microsoft Fortran PowerStation 集成环境 MAT 文件应用程序的建立和调试

出于同样的目的，即为了简化整个 FORTRAB 语言 MAT 文件应用程序的建立和调试过程，我们将给出一种在 Microsoft Fortran PowerStation 集成环境中完成整个程序编译和调试过程的方法。其具体步骤如下：

第一步，进入 Microsoft Fortran PowerStation 集成环境，从下拉式菜单 File 选取 New 菜单项，系统将弹出一个如图 5.4 所示的选项框，选择其中的“Project Workspace”项，并点击 OK 按钮，用于创建一个新的项目空间；

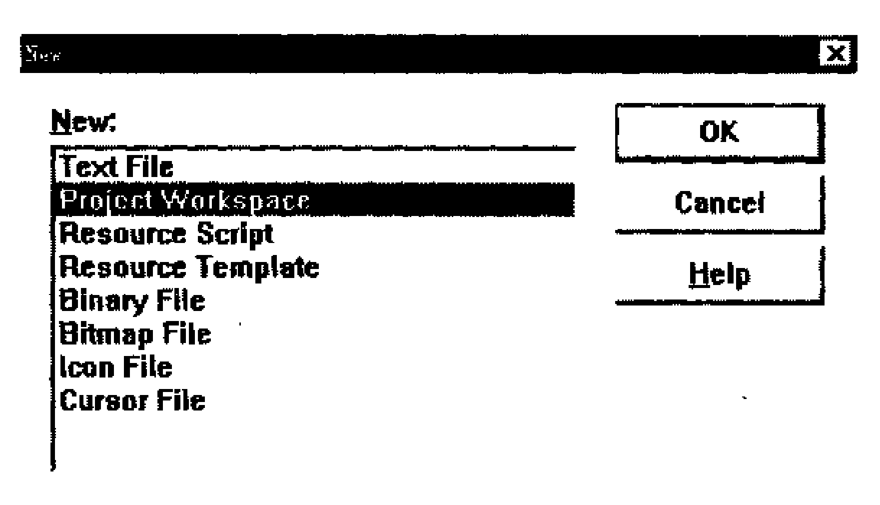


图 5.4　New 选项框

第二步，在关闭 New 选项框之后，系统将弹出 New Project Workspace 选项框，如图 5.5 所示，用户可以选择其中多种类型的项目用于创建 MAT 文件应用程序，例如可以为 Application 或 Console Application，也可以为 QuickWin Application 或 Standard Graphics Application，为了说明方便，这里我们使用 Console Application 类型。点取 Type 选项框中的 Console Application 项，并在 Name 编辑框中输入项目名字后，单击 Create 按钮，创建项目；

第三步，在项目工程创建完毕之后，选择下拉式菜单 Tools 中的菜单项 Options，将弹出 Options 对话框，选择其中的 Directories 属性页，如图 5.6 所示，在其中的 Show directories for 下拉式选项框中分别选择 Include Files 和 Library Files，在下部的编辑框中输入以下路径：

图 5.5　New Project Workspace 选项框

图 5.6　Options 对话框

MATLAB 根目录\EXTERN\INCLUDE

MATLAB 根目录\EXTERN\LIB

然后选择 OK 按钮；

第四步，在 DOS 命令框状态下，进入用户安装 Microsoft FORTRAN PowerStation 的目录，如 d：\msdev，并且进入该目录下的子目录\bin，按下面的格式运行该目录下的命令 lib：

```
lib /def:%MATLAB%\extern\include\dfmx.def /machine: ix86 /OUT: LIB_NAME1.lib /NOLOGO
lib /def:%MATLAB%\extern\include\dfmat.def /machine: ix86 /OUT: LIB_NAME2.lib /
```

NOLOGO

lib /def:%MATLAB%\extern\include\dfeng.def /machine: ix86 /OUT: LIB_NAME3.lib /NOLOGO

以产生三个分别名为 LIB_NAME1.lib，LIB_NAME2.lib 和 LIB_NAME3.lib 的静态链接库文件，其中%MATLAB%表示用户安装 MATLAB 的根目录，如 d:\matlab，一旦库文件成功生成后，可以应用在所有的 MAT 文件应用程序的工程之中，而无须重复生成；

第五步，选择下拉式菜单 Insert 的菜单项 Files into Project，选择库文件第四步中生成的三个 lib 文件，将它们嵌入到当前的工程中；同时选择用户的 MAT 文件应用程序的源文件，将其也嵌入到当前的工程中。

完成了以上五步工作之后，用户就可以在 Microsoft Fortran PowerStation 集成环境中对 MAT 文件应用程序进行编译和调试了。对于其他项目类型的 MAT 文件应用程序，步骤大致相同。

5.4 MAT 文件库函数说明

本节将分别基于 C 语言和 FORTRAN 语言，对 MAT 文件函数库中的函数使用进行说明。

5.4.1 C 语言 MAT 文件函数的使用说明

在 MAT 文件函数库中，总共提供了 19 个 C 语言的引擎函数，它们的声明分别如下：

```
int matClose (MATFile *mfp)
int matDeleteArray (MATFile *mfp, const char *name)
matDeleteMatrix (Obsolete)
mxArray *matGetArray (MATFile *mfp, const char *name)
mxArray *matGetArrayHeader (MATFile *mfp, const char *name)
char **matGetDir (MATFile *mfp, int *num)
FILE *matGetFp (MATFile *mfp)
matGetFull (Obsolete)
matGetMatrix (Obsolete)
mxArray *matGetNextArray (MATFile *mfp)
mxArray *matGetNextArrayHeader (MATFile *mfp)
matGetNextMatrix (Obsolete)
matGetString (Obsolete)
MATFile *matOpen (const char *filename, const char *mode)
int matPutArray (MATFile *mfp, const mxArray *mp)
int matPutArrayAsGlobal (MATFile *mfp, const mxArray *mp)
matPutFull (Obsolete)
matPutMatrix (Obsolete)
matPutString (Obsolete)
```

所有这些函数均在头文件 mat.h 中予以声明，在使用它们时，必须对该头文件进行

包含。下面我们对这些函数进行逐一说明。

1. matClose

功　能：用于关闭一个 MAT 文件。

语　法：#include "mat.h"

int matClose (MATFile *mfp);

说　明：函数 matClose 用于关闭一个已经打开的 MAT 文件。函数仅有一个输入参数 mfp，为一个 MATFile 类型的指针，指向开启的 MAT 文件。如果函数执行成功，其返回值为 0；否则将返回一个写文件错误。

举　例：文件 matcreat.c 是 MATLAB 提供的一个范例程序，位于目录

MATLAB 根目录\extern\examples\eng_mat\

中，它对函数 matClose，matPutArray，matGetArray，matPutArrayAsGlobal 和 matOpen 的使用给出了样例，其源代码如下，我们将在程序中对各语句的功能加以注释：

```
/* 头文件包含，注意其中对头文件 mat.h 的包含，该文件中包含了对所有 MAT
   文件库函数的说明 */
#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "mat.h"
#define BUFSIZE 255
/* 子函数 create 的定义，其功能为创建一个 MAT 文件，并完成一定的读写操作
   */
int create (const char *file)
{
  /* 变量声明，其中变量 pmat 为一个 MAT 文件指针，pa1、pa2 和 pa3 分别为
     指向 mxArray 结构体的指针 */
  MATFile *pmat;
  mxArray *pa1, *pa2, *pa3;
  double data [9] = { 1.0, 4.0, 7.0, 2.0, 5.0, 8.0, 3.0, 6.0, 9.0 };
  char str [BUFSIZE];
  /* 输出 MAT 文件名 */
  printf ("Creating file %s...\n\n", file);
  /* 使用函数 matOpen 建立一个新的 MAT 文件，如果已经存在同名的 MAT 文
     件，则将原来的内容删除，有关函数 matOpen 的内容将在后面说明 */
  pmat = matOpen (file, "w");
  /* 判断 MAT 文件是否开启成功 */
  if (pmat == NULL)
  {
     printf ("Error creating file %s\n", file);
     printf (" (do you have write permission in this directory?) \n");
     return (1);
```

```
}
/* 创建了三个 mxArray 结构体对象，其中 pa1 和 pa2 为 3×3 的双精度实型矩
   阵，并且分别命名为 LocalDouble 和 GlobalDouble，并对 pa2 进行了赋值；
   pa3 被声明为一个字符串类型的阵列，并进行了赋值 */
pa1 = mxCreateDoubleMatrix (3, 3, mxREAL);
mxSetName (pa1, "LocalDouble");
pa2 = mxCreateDoubleMatrix (3, 3, mxREAL);
mxSetName (pa2, "GlobalDouble");
memcpy ( (void * )  (mxGetData (pa2)), (void * ) data, 3 * 3 * sizeof
(double));
pa3 = mxCreateString ("MATLAB: the language of technical
computing");
mxSetName (pa3, "LocalString");
/* 使用函数 matPutArray 将三个 mxArray 结构体变量 pa1、pa2 和 pa3 输送到
   MAT 文件中，函数 matPutArray 将在后面详细说明 */
matPutArray (pmat, pa1);
matPutArrayAsGlobal (pmat, pa2);
matPutArray (pmat, pa3);
/* 为结构体变量 pa1 赋值，再次将其输出到同一个 MAT 文件中 */
memcpy ( (char * )  (mxGetPr (pa1)), (char * ) data, 3 * 3 * sizeof
(double));
matPutArray (pmat, pa1);
/* 删除三个结构体变量 pa1、pa2 和 pa3 */
mxDestroyArray (pa1);
mxDestroyArray (pa2);
mxDestroyArray (pa3);
/* 关闭 MAT 文件，并进行判断，是否关闭成功 */
if (matClose (pmat) ! = 0)
{
    printf ("Error closing file %s\n", file);
    return (1);
}
/* 再次开启 MAT 文件 */
pmat = matOpen (file, "r");
if (pmat == NULL)
{
    printf ("Error reopening file %s\n", file);
    return (1);
}
/* 将前面写入的数据使用函数 matGetArray 读取，函数 matGetArray 将在后
   面进行说明 */
pa1 = matGetArray (pmat, "LocalDouble");
if (pa1 == NULL)
```

```
{
    printf ("Error reading existing matrix LocalDouble\n");
    return (1);
}
if (mxGetNumberOfDimensions (pa1) ! = 2)
{
    printf ("Error saving matrix: result does not have two
    dimensions\n");
    return (1);
}
pa2 = matGetArray (pmat, "GlobalDouble");
if (pa2 == NULL)
{
    printf ("Error reading existing matrix LocalDouble\n");
    return (1);
{
if (! (mxIsFromGlobalWS (pa2)))
{
    printf ("Error saving global matrix: result is not global\n");
    return (1);
}
pa3 = matGetArray (pmat, "LocalString");
if (pa3 == NULL)
{
    printf ("Error reading existing matrix LocalDouble\n");
    return (1);
}
mxGetString (pa3, str, BUFSIZE);
if (strcmp (str, "MATLAB: the language of technical computing"))
{
    printf ("Error saving string: result has incorrect
    contents\n");
    return (1);
}
/* 删除结构体变量 */
mxDestroyArray (pa1);
mxDestroyArray (pa2);
mxDestroyArray (pa3);
if (matClose (pmat) ! = 0)
{
    printf ("Error closing file %s\n", file);
    return (1);
}
printf ("Done\n");
return (0);
```

```
}
/* 主程序 */
int main ()
 {
    int result;
    result = create ("mattest.mat");
    return (result==0)? EXIT_SUCCESS: EXIT_FAILURE;
 }
```

对该程序编译后执行，可以得到一个名为 mattest.mat 的 MAT 文件，在 MATLAB 中使用下面的命令可以对文件的内容进行查阅，如下：

```
? whos -file mattest.mat
Name Size Bytes Class
GlobalDouble    3x3    72 double array (global)
LocalDouble     3x3    72 double array
LocalString     1x43   86 char array
```

2. matDeleteArray

功　能：从 MAT 文件中删除一个阵列。

语　法：#include "mat.h"

int matDeleteArray (MATFile *mfp, const char *name);

说　明：函数 matDeleteArray 允许用户删除已开启 MAT 文件中指定名字的阵列，其输入参数 mfp 代表了一个开启的 MAT 文件，而 name 为一个希望删除的阵列名。如果函数执行成功，将返回 0，否则将返回一个非零值。

举　例：假设存在一个已经开启的 MAT 文件 mfp，并且在该文件中存在一个名为 data 的阵列，通过下面的语句可以将其从文件中删除：

```
int status;
status = matDeleteArray (mfp, "data");
if (status != 0)
{
    printf ("Error delete Array");
}
```

3. matDeleteMatrix (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MAT 文件应用程序中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 matDeleteArray 来替代。如果需要使用已经存在对函数 matDeleteMatrix 调用的 MAT 文件应用程序，而不希望对源程序进行修改，那么在对这类 MAT 文件应用程序进行编

译时，必须使用mex命令参数-V4，声明编译为与MATLAB V4.0版本兼容的MAT文件应用程序。

4. matGetArray

功　能：从MAT文件中读取数据。

语　法：#include "mat.h"

mxArray *matGetArray (MATFile *mfp, const char *name);

说　明：使用函数matGetArray允许用户从一个开启的MAT文件中读取阵列数据，其输入参数的含义分别如下：mfp代表了一个已经开启的MAT文件，为一个MATFile类型的指针，name为一个字符指针，代表了用户希望提取阵列的名字。如果函数执行成功，函数将新初始化一个mxArray结构体对象，然后将MAT文件中指定名字阵列的数据拷贝到新分配的mxArray结构体对象中，并返回指向该对象的指针；如果函数执行失败，将返回NULL。这里必须非常注意一点，在程序结束前必须将函数产生的mxArray对象删除，否则将导致内存的泄漏。

举　例：参见函数matClose中的范例程序matcreat.c。

5. matGetArrayHeader

功　能：获取MAT文件中某个阵列的信息。

语　法：#include "mat.h"

mxArray *matGetArrayHeader (MATFile *mfp, const char *name);

说　明：使用函数matGetArrayHeader允许用户从一个开启的MAT文件中读取指定阵列的信息，信息中包括了除阵列的pr、pi、ir和jc信息外的所有信息，有关pr、pi、ir和jc的描述请读者参见第二章。函数的输入参数的含义分别如下：mfp代表了一个已经开启的MAT文件，为一个MATFile类型的指针，name为一个字符指针，代表了用户希望提取信息的阵列的名字。这里必须注意的一点是通过函数matGetArrayHeader提取出的阵列信息，仅仅用于使用，而不能被写回MAT文件或传送到MATLAB中。如果函数执行成功，将返回一个指向mxArray结构体对象的指针，在所指向的mxArray结构体对象中，pr、pi、ir和jc这些信息均被设置为-1，不管原来为何值。

举　例：文件matdgns.c是MATLAB提供的一个范例程序，位于目录

MATLAB根目录\extern\examples\eng_mat\

中，它对函数matClose，matGetDir，matGetNextArray，matGetNextArrayHeader和matOpen的使用给出了样例，其源代码如下，我们将在程序中对各语句的功能加以注释：

```
/* 头文件包含，注意其中对头文件mat.h的包含，该文件中包含了对所有MAT
   文件库函数的说明 */
#include <stdio.h>
#include <stdlib.h>
```

```
#include "mat.h"
/* 子函数 create 的定义，其功能为 */
int diagnose (const char *file)
{
/* 变量声明 */
MATFile *pmat;
char     **dir;
int     ndir;
int    i;
mxArray *pa;
/* 输出 MAT 文件名 */
printf ("Reading file %s...\n\n", file);
/* 使用函数 matOpen 开启 MAT 文件，并判断是否成功 */
pmat = matOpen (file, "r");
if (pmat == NULL)
{
   printf ("Error opening file %s\n", file);
   return (1);
}
/* 通过函数 matGetDir 得到 MAT 文件中阵列的数据，函数 matGetDir 将在后续
   内容中说明 */
dir = matGetDir (pmat, &ndir);
/* 通过函数 matGetDir 的返回值判断是否执行成功 */
if (dir == NULL)
{
   printf ("Error reading directory of file %s\n", file);
   return (1);
} else
{
   printf ("Directory of %s: \n", file);
   for (i=0; i < ndir; i++)
     printf ("%s\n", dir [i]);
}
mxFree (dir);
/* 关闭 MAT 文件 */
if (matClose (pmat) ! = 0)
{
   printf ("Error closing file %s\n", file);
   return (1);
}
/* 重新开启 MAT 文件 */
pmat = matOpen (file, "r");
if (pmat == NULL)
```

```
{
  printf ("Error reopening file %s\n", file);
  return (1);
}
/* 通过循环过程和 matGetNextArrayHeader 获取所有的 MAT 文件阵列的头信
   息 */
printf ("\nExamining the header for each variable: \n");
for (i=0; i < ndir; i++)
{
  pa = matGetNextArrayHeader (pmat);
  if (pa == NULL)
   {
    printf ("Error reading in file %s\n", file);
    return (1);
   }
  /* 分析头信息 */
 printf ("According to its header, array %s has %d dimensions\n",
      mxGetName(pa), mxGetNumberOfDimensions(pa));
  if (mxIsFromGlobalWS (pa))
    printf (" and was a global variable when saved\n");
  else
    printf (" and was a local variable when saved\n");
  mxDestroyArray (pa);
}
/* 关闭 MAT 文件 */
if (matClose (pmat) != 0)
{
  printf ("Error closing file %s\n", file);
  return (1);
}
/* 再次开启 MAT 文件 */
pmat = matOpen (file, "r");
if (pmat == NULL)
{
  printf ("Error reopening file %s\n", file);
  return (1);
  }
/* 通过循环过程和函数 matGetNextArray 获得所有的 MAT 文件阵列 */
printf ("\nReading in the actual array contents: \n");
for (i=0; i<ndir; i++)
{
    pa = matGetNextArray (pmat);
    if (pa == NULL) {
    printf ("Error reading in file %s\n", file);
```

```
        return (1);
    }
    /* 分析阵列 */
    printf ("According to its contents, array %s has %d dimensions\n",
    mxGetName (pa), mxGetNumberOfDimensions (pa));
    if (mxIsFromGlobalWS (pa))
    {   printf (" and was a global variable when saved\n");
    else
         printf (" and was a local variable when saved\n");
    mxDestroyArray (pa);
    }
    /* 关闭 MAT 文件 */
    if (matClose (pmat) ! = 0)
    {
         printf ("Error closing file %s\n", file);
         return (1);
    }
    printf ("Done\n");
    return (0);
}
/* 主函数 */
int main (int argc, char * * argv)
 {
   int result;
   if (argc > 1)
      result = diagnose (argv [1]);
   else
   {
      result = 0;
      printf ("Usage: matdgns <matfile>");
      printf (" where <matfile> is the name of the MAT-file");
      printf (" to be diagnosed\n");
   }
   return (result==0)? EXIT_SUCCESS: EXIT_FAILURE;
 }
```

对该 MAT 文件应用程序编译后，执行可得如下的结果，其中 mattest.mat 为前面 matcreat.c 程序生成的 MAT 文件：

```
E: \MATLAB\extern\examples\eng_mat>matdgns mattest.mat
Reading file mattest.mat...
Directory of mattest.mat:
GlobalDouble
LocalString
```

```
LocalDouble

Examining the header for each variable:
According to its header, array GlobalDouble has 2 dimensions
    and was a global variable when saved
According to its header, array LocalString has 2 dimensions
    and was a local variable when saved
According to its header, array LocalDouble has 2 dimensions
    and was a local variable when saved

Reading in the actual array contents:
According to its contents, array GlobalDouble has 2 dimensions
    and was a global variable when saved
According to its contents, array LocalString has 2 dimensions
    and was a local variable when saved
According to its contents, array LocalDouble has 2 dimensions
    and was a local variable when saved
Done
```

6. matGetDir

功　能：获得一个 MAT 文件中的所有阵列的目录。

语　法：#include "mat.h"

char * * matGetDir (MATFile * mfp, int * num);

说　明：函数 matGetDir 允许用户从一个开启的 MAT 文件中获取所有阵列的名字。如果函数执行成功将返回一个字符指针数组，数组的每一个元素均为字符指针，指向一个以空字符 NULL 结尾的字符串，该字符串代表了 MAT 文件中某个阵列的名字，同时必须说明的一点是该数组通过函数 mxCalloc 分配内存，在程序结束之前，必须使用函数 mxFree 进行释放，否则将导致内存泄漏；数组元素的个数存储在整型指针 num 所指向的内存区域中。如果函数执行失败，返回参数 num 中的值将为－1，并且返回一个空指针数组。如果返回参数 num 中的值为 0，表示 MAT 文件中没有包含任何阵列。

举　例：参见函数 matGetArrayHeader 中的例子 matdgna.c。

7. matGetFp

功　能：获得 MAT 文件的 C 语言文件句柄。

语　法：#include "mat.h"

FILE * matGetFp (MATFile * mfp);

说　明：通过函数 matGetFp，用户可以获得一个指向已经处于开启状态的 MAT 文件的 C 语言文件句柄，通过这个句柄，用户就可以方便地使用 C 语言提供的库函数 feof 或 ferror 来判断错误的原因。该函数仅有一个输入参数，即表示处于开启状态的 MAT 文件的 MATFile 类型的指针 mfp。

举　例：程序是一个使用函数 matGetFp 的范例程序，其源代码如下，我们将在程序

中附加注释：

```
/* 头文件包含，注意其中对头文件mat.h的包含，该文件中包含了对所有MAT文件
   库函数的说明 */ */
#include <stdio.h>
#include <stdlib.h>
#include "mat.h"
/* 子函数getfp，用于开启一个MAT文件，并依次读取其中的阵列 */
int getfp (const char *file)
{
  /* 变量声明 */
  MATFile *pmat;
  int i;
  mxArray *pa;
  printf ("Reading file %s...\n\n", file);
  /* 开启MAT文件 */
  pmat = matOpen (file, "r");
  if (pmat == NULL)
  {
      printf ("Error opening file %s\n", file);
      return (1);
  }
  /* 使用函数matGetNextArray 依次读取MAT文件中的阵列 */
  printf ("\nReading in the actual array contents: \n");
  for (i=0; ; i++)
  {
      pa = matGetNextArray (pmat);
      if (pa == NULL)
      {
        /* 获取C语言MAT文件句柄 */
        FILE *File = matGetFp (pmat);
        /* 使用C语言库函数对错误原因进行判断，并分别进行不同的处理 */
        if ( feof (File) > 0 )
          {
          printf ("End of file %s\n", file);
          break;
        }
        if ( ferror (File) > 0 )
        {
          printf ("Error of reading %s\n", file);
          return (1);
        }
      }
      /* 分析阵列pa */
```

```
        printf("According to its contents, array %s has %d dimensions\n",
           mxGetName (pa), mxGetNumberOfDimensions (pa));
        if (mxIsFromGlobalWS (pa))
             printf (" and was a global variable when saved\n");
        else
             printf (" and was a local variable when saved\n");
        mxDestroyArray (pa);
    }
    /* 关闭 MAT 文件 */
    if (matClose (pmat) ! = 0)
    {
        printf ("Error closing file %s\n", file);
        return (1);
    }
    printf ("Done\n");
    return (0);
}
/* 主函数 */
int main (int argc, char * * argv)
 {
  int result;
  if (argc > 1)
     result = diagnose (argv [1]);
  else
  {
     result = 0;
     printf ("Usage: matdgns <matfile>");
     printf (" where <matfile> is the name of the MAT-file");
     printf (" to be diagnosed\n");
  }
  return (result==0)? EXIT_SUCCESS: EXIT_FAILURE;
 }
```

对该程序编译后执行，将显示如下内容，其中 mattest. mat 为前面使用程序 matcreat 所创建的 MAT 文件：

```
E: \MATLAB\extern\examples\eng_mat>matgetfp mattest. mat
Reading file mattest. mat...

Reading in the actual array contents:
According to its contents, array GlobalDouble has 2 dimensions
   and was a global variable when saved
According to its contents, array LocalString has 2 dimensions
   and was a local variable when saved
According to its contents, array LocalDouble has 2 dimensions
```

```
    and was a local variable when saved
End of file mattest.mat
Done
```

8. matGetFull (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MAT 文件应用程序中，否则在编译时系统将报错。如果需要使用已经存在对函数 matGetFull 调用的 MAT 文件应用程序，而不希望对源程序进行修改，那么在对这类 MAT 文件应用程序进行编译时，必须使用 mex 命令参数 -V4，声明编译为与 MATLAB V4.0 版本兼容的 MAT 文件应用程序。

用户可以通过自定义该函数完成同样的功能，源代码如下，其中使用了与函数 matGetFull 功能相似的函数 matGetArray：

```
int matGetFull (MATFile *fp, char *name, int *m, int *n,
                double **pr, double **pi)
{
    mxArray *parr;
    /* 获取阵列 */
    parr = matGetArray (fp, name);
    if (parr == NULL)
        return (1);
    if (! mxIsDouble (parr))
    {
        mxDestroyArray (parr);
        return (1);
    }
    /* 获得阵列的信息 */
    *m = mxGetM (parr);
    *n = mxGetN (parr);
    *pr = mxGetPr (parr);
    *pi = mxGetPi (parr);
    /* 将阵列的数据指针置空，以便于删除对象 */
    mxSetPr (parr, (void *) 0);
    mxSetPi (parr, (void *) 0);
    mxDestroyArray (parr);
    return (0);
}
```

9. matGetMatrix (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保

持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MAT 文件应用程序中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 matGetArray 来替代。如果需要使用已经存在对函数 matGetMatrix 调用的 MAT 文件应用程序，而不希望对源程序进行修改，那么在对这类 MAT 文件应用程序进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MAT 文件应用程序。

10. matGetNextArray

功　能：读取 MAT 文件中的下一个阵列。

语　法：
```
#include "mat.h"
mxArray *matGetNextArray (MATFile *mfp);
```

说　明：函数 matGetNextArray 的功能为读取输入参数 mfp 指向的处于开启状态的 MAT 文件中的下一个阵列的数据，并将返回一个新分配的 mxArray 结构体的指针，其中输入参数 mfp 为一个 MATFile 类型的指针。通过该函数，用户可以通过一个循环过程，依次访问 MAT 文件中所有的阵列。不过必须注意的一点是，对函数 matGetNextArray 的使用必须紧接在函数 matOpen 之后，也就是说在两者之间不能加入其他的 MAT 文件操作函数，否则函数 matGetNextArray 将不知道“下一个（next）”的定义。如果到达文件结尾或者函数执行出错，函数将返回一个 NULL 指针，这时可以通过标准的 C 语言函数 feof 和 ferror 进行判断是哪一种情况。在程序结束之前，必须将函数 matGetNextArray 分配的 mxArray 结构体删除，否则将导致内存泄漏。

举　例：参见函数 matGetArrayHeader 中的例子 matdgna.c。

11. matGetNextArrayHeader

功　能：获取 MAT 文件中下一个阵列的信息。

语　法：
```
#include "mat.h"
mxArray *matGetNextArrayHeader (MATFile *mfp);
```

说　明：该函数的功能和用法与函数 matGetNextArray 大致相同，不过它只读取 MAT 文件中阵列的信息。

举　例：参见函数 matGetArrayHeader 中的例子 matdgna.c。

12. matGetNextMatrix (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MAT 文件应用程序中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 matGetNextArray 来替代。如果需要使用已经存在对函数 matGetNextMatrix 调用的 MAT 文件应用程序，而不希望对源程序进行修改，那么在对这类 MAT 文件应用程序进行

编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MAT 文件应用程序。

13. matGetString (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MAT 文件应用程序中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数以下的语句来替代：

```
#include "mat.h"
#include "matrix.h"
mxArray *matGetArray (MATFile *mfp, const char *name);
int mxGetString (const mxArray *array_ptr, char *buf, int buflen)
```

如果需要使用已经存在对函数 matGetString 调用的 MAT 文件应用程序，而不希望对源程序进行修改，那么在对这类 MAT 文件应用程序进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MAT 文件应用程序。

14. matOpen

功　能：开启一个 MAT 文件。

语　法：
```
#include "mat.h"
MATFile *matOpen (const char *filename, const char *mode);
```

说　明：通过函数 matOpen，用户可以开启 MAT 文件用于读写。如果函数执行成功将返回一个 MATFile 类型的指针，否则返回值为 NULL。函数拥有两个输入参数，其中 filename 为一个指向字符串的指针，字符串中包含了希望开启的 MAT 文件的文件名，mode 为一个字符指针，用来说明开启 MAT 文件的类型，它有以下几种取值：

- "r"，表示使用只读方式开启 MAT 文件；
- "u"，表示以可读和可写方式开启文件，即可以对文件中的某一些阵列进行更新；
- "w"，表示以只写方式开启文件，如果磁盘中存在同名的 MAT 文件，那么以这种方式开启 MAT 文件将删除原有文件中的所有内容；如果原来不存在同名的 MAT 文件，那么将创建一个新的文件；
- "w4"，表示创建一个 MATLAB 4.X 版本的 MAT 文件。

举　例：参见函数 matGetArrayHeader 中的例子 matdgna.c 和函数 matClose 中的范例程序 matcreat.c。

15. matPutArray

功　能：将阵列写入到 MAT 文件中。

语　法：#include "mat.h"

int matPutArray (MATFile *mfp, const mxArray *mp);

说　明：通过MAT函数，用户可以将一个mxArray结构体mp输入到参数mfp所表示的MAT文件中。如果在原来的MAT文件中存在同名的mxArray结构体，那么将覆盖旧的mxArray结构体；如果不存在同名的结构体，那么将在文件的末尾添加结构体。这里需要说明的一点是新老结构体的大小可以不相同。如果函数执行成功，将返回0，否则将返回一个非零值，这时可以使用C语言的库函数feof或ferror来判断错误的原因。

举　例：参见函数matClose中的范例程序matcreat.c。

16. matPutArrayAsGlobal

功　能：将阵列写入到MAT文件中。

语　法：#include "mat.h"

int matPutArrayAsGlobal (MATFile *mfp, const mxArray *mp);

说　明：函数matPutArrayAsGlobal的功能与函数matPutArray大致相同，惟一的区别就是用matPutArrayAsGlobal写入MAT文件的阵列在读取时，将存放入MATLAB的全局工作空间。

举　例：参见函数matClose中的范例程序matcreat.c。

17. matPutFull (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同MATLAB V4.0版本保持兼容。一般情况下，该函数不允许出现在MATLAB V5.0版本以上的MAT文件应用程序中，否则在编译时系统将报错。如果需要使用已经存在对函数matPutFull调用的MAT文件应用程序，而不希望对源程序进行修改，那么在对这类MAT文件应用程序进行编译时，必须使用mex命令参数-V4，声明编译为与MATLAB V4.0版本兼容的MAT文件应用程序。

用户可以通过自定义该函数完成同样的功能，源代码如下，其中使用了与函数matGetFull功能相似的函数matPutArray：

```
int matPutFull (MATFile *ph, char *name, int m, int n,
                double *pr, double *pi)
{
    int retval;
    mxArray *parr;
    /* 创建一个新的双精度矩阵 */
    parr = mxCreateDoubleMatrix (0, 0, 0);
      if (parr==NULL)
        return (1);
    /* 设置矩阵的信息 */
    mxSetM (parr, m);
    mxSetN (parr, n);
    mxSetName (parr, name);
```

```
        mxSetPr (parr, pr);
        mxSetPi (parr, pi);
        /* 将矩阵输出到 MAT 文件中 */
        retval = matPutArray (ph, parr);
        /* 删除矩阵，释放内存 */
        mxSetPr (parr, (void * ) 0);
        mxSetPi (parr, (void * ) 0);
        mxDestroyArray (parr);
        return (retval);
    }
```

18. matPutMatrix (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MAT 文件应用程序中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数 matPutArray 来替代。

如果需要使用已经存在对函数 matPutMatrix 调用的 MAT 文件应用程序，而不希望对源程序进行修改，那么在对这类 MAT 文件应用程序进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MAT 文件应用程序。

19. matPutString (*Obsolete*)

说　明：该函数为一个过时的函数，保留它的目的是为了同 MATLAB V4.0 版本保持兼容。一般情况下，该函数不允许出现在 MATLAB V5.0 版本以上的 MAT 文件应用程序中，否则在编译时系统将报错。在 MATLAB V5.0 以上的版本中，为了完成同样的功能，可以使用函数以下的语句来替代：

```
#include "matrix.h"
#include "mat.h"
mxArray *mxCreateString (char *str)
int matPutArray (MATFile *mfp, const mxArray *mp);
void mxDestroyArray (mxArray *array_ptr)
```

如果需要使用已经存在对函数 matPutString 调用的 MAT 文件应用程序，而不希望对源程序进行修改，那么在对这类 MAT 文件应用程序进行编译时，必须使用 mex 命令参数-V4，声明编译为与 MATLAB V4.0 版本兼容的 MAT 文件应用程序。

5.4.2 FORTRAN 语言 MAT 文件函数的使用说明

在 MAT 文件函数库中，总共提供了 11 个 FORTRAN 语言的引擎函数，它们的声明分别如下：

```
integer * 4 function matClose (mfp)
subroutine matDeleteMatrix (mfp, name)
integer * 4 function matGetDir (mfp, num)
integer * 4 function matGetFull (mfp, name, m, n, pr, pi)
integer * 4 function matGetMatrix (mfp, name)
integer * 4 function matGetNextMatrix (mfp)
integer * 4 function matGetString (mfp, name, str, strlen)
integer * 4 function matOpen (filename, mode)
integer * 4 function matPutFull (mfp, name, m, n, pr, pi)
integer * 4 function matPutMatrix (mfp, mp)
integer * 4 function matPutString (mfp, name, str)
```

下面我们对这些函数进行逐一说明。

1. matClose

功　能：关闭 MAT 文件。

语　法：integer * 4 function matClose (mfp)

integer * 4 mfp

说　明：函数 matClose 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。该函数的功能为关闭一个已经打开的 MAT 文件。函数仅有一个输入参数 mfp，为一个 MATFile 类型的指针，指向开启的 MAT 文件。如果函数执行成功，其返回值为 0；否则将返回一个写文件错误。

举　例：matdemo1.f 是 MATLAB 提供的一个范例程序，位于目录

MATLAB 根目录\extern\examples\eng _ mat\

中，它对函数 matClose，matGetMatrix，matPutMatrix，matDeleteMatrix 和 matOpen 的使用给出了样例，其源代码如下，我们将在程序中对各语句的功能加以注释：

```
      program matdemo1
C     对程序中使用变量和函数子程序的类型进行声明
      integer matOpen, mxCreateFull, mxCreateString
      integer matGetMatrix, mxGetPr
      integer mp, pa1, pa2, pa3
      integer status, matClose
      double precision dat (9)
      data dat / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 /
C     使用函数 matOpen 开启 MAT 文件，并判断函数是否执行成功
      write (6, * ) 'Creating MAT-file matdemo.mat ...'
      mp = matOpen ('matdemo.mat', 'w')
      if (mp .eq. 0) then
         write (6, * ) 'Can't open "matdemo.mat" for writing.'
```

```
        write (6, *) ' (Do you have write permission in this directory?)'
        stop
    end if
C   创建输入到 MAT 文件中的 mxArray 结构体变量 pa1
    pa1 = mxCreateFull (3, 3, 0)
    call mxSetName (pa1, 'Numeric')
C   创建输入到 MAT 文件中的 mxArray 结构体变量 pa2
    pa2 = mxCreateString ('MATLAB: The language of computing')
    call mxSetName (pa2, 'String')
C   创建输入到 MAT 文件中的 mxArray 结构体变量 pa3
    pa3 = mxCreateString ('MATLAB: The language of computing')
    call mxSetName (pa3, 'String2')
C   调用子例行程序 matPutMatrix 将 mxArray 结构体变量 pa1、pa2
C   和 pa3 输入到 MAT 文件中
    call matPutMatrix (mp, pa1)
    call matPutMatrix (mp, pa2)
    call matPutMatrix (mp, pa3)
C   给 mxArray 结构体赋值，并重新将 pa1 输入到 MAT 文件中
    call mxCopyReal8ToPtr (dat, mxGetPr (pa1), 9)
    call matPutMatrix (mp, pa1)
C   调用子例行程序 matDeleteMatrix 从 MAT 文件中删除名为 String2
C   的 mxArray 结构体变量
    call matDeleteMatrix (mp, 'String2')
C   使用函数 matClose 关闭 MAT 文件
    status = matClose (mp)
    if (status .ne. 0) then
        write (6, *) 'Error closing MAT-file'
        stop
    end if
C   再次开启 MAT 文件，并判断函数是否执行成功
    mp = matOpen ('matdemo.mat', 'r')
    if (status .ne. 0) then
        write (6, *) 'Can't open "matdemo.mat" for reading.'
        stop
    end if
C   使用函数子程序 matGetMatrix 从 MAT 文件中读取阵列
    pa1 = matGetMatrix (mp, 'Numeric')
    if (mxIsNumeric (pa1) .eq. 0) then
        write (6, *) 'Invalid non-numeric matrix written to MAT-file'
        stop
    end if
C
```

```
      pa2 = matGetMatrix (mp, 'String')
      if (mxIsString (pa2) .eq. 0) then
          write (6, * ) 'Invalid non-numeric matrix written to MAT-file'
          stop
      end if
C
      pa3 = matGetMatrix (mp, 'String2')
      if (pa3 .ne. 0) then
          write (6, * ) 'String2 not deleted from MAT-file'
          stop
      end if
C     释放内存
      call mxFreeMatrix (pa1)
      call mxFreeMatrix (pa2)
      call mxfreeMatrix (pa3)
C     关闭 MAT 文件
      status = matClose (mp)
      if (status .ne. 0) then
          write (6, * ) 'Error closing MAT-file'
          stop
      end if
C
      write (6, * ) 'Don't creating MAT-file'
      stop
      end
```

对该程序编译后执行将创建一个名为 matdemo.mat 的 MAT 文件，在 MATLAB 命令提示符下键入如下命令可以查看文件的内容：

```
? whos -file matdemo.mat
Name Size Bytes Class
Numeric 3x3 72 double array
String 1x33 66 char array
Grand total is 42 elements using 138 bytes
```

2. matDeleteMatrix

功　能：从 MAT 文件中删除一个阵列。

语　法：subroutine matDeleteMatrix (mfp, name)

integer * 4 mfp

character * (*) name

说　明：该函数为一个 FORTRAN 语言子例行程序，通过该函数，用户可以从一个开启的 MAT 文件中删除一个指定名字的阵列。输入参数 mfp 为一个指向 MAT 文件的 MATFile 类型的指针；name 为一个字符串变量，包含了希望删除的阵列名。如果函数执行成功，返回值为零，否则返回值为非零。

举　例：参见函数子程序 matClose 的范例程序 matdemo1. f。

3. matGetDir

功　能：获得 MAT 文件中所有阵列的目录。

语　法：integer * 4 function matGetDir (mfp, num)

integer * 4 mfp, num

说　明：函数 matClose 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。通过该函数，用户可以获取一个处于开启状态的 MAT 文件中的所有阵列的目录。如果函数执行成功将返回一个指针，指向一个内部数组，数组的每一个元素均为指向 MAT 文件中阵列名字的指针，同时必须说明的一点是该内部数组通过函数 mxCalloc 分配内存，在程序结束之前，必须使用函数 mxFree 进行释放，否则将导致内存泄漏；数组元素的个数存储在指针 num 所指向的内存区域中。如果函数执行失败，返回参数 num 中的值将为-1，并且返回一个空指针数组。如果返回参数 num 中的值为 0，表示 MAT 文件中没有包含任何阵列。

举　例：matdemo2. f 是 MATLAB 提供的一个范例程序，位于目录

MATLAB 根目录\extern\examples\eng _ mat\

中，它对函数 matClose，matGetDir，matGetNextMatrix 和 matOpen 的使用给出了样例，其源代码如下，我们将在程序中对各语句的功能加以注释：

```
      program matdemo2
C     对程序中使用变量和函数子程序的类型进行声明
      integer matOpen, matGetDir, matGetNextMatrix
      integer mp, dir, adir (100), pa
      integer mxGetM, mxGetN, matClose
      integer ndir, i, stat
      character * 32 names (100), name, mxGetName
C     使用函数 matOpen 开启 MAT 文件 matdemo. mat，并判断
C     函数是否执行成功
      mp = matOpen ('matdemo. mat', 'r')
      if (mp .eq. 0) then
         write (6, * ) 'Can''t open "matdemo. mat".'
         stop
      end if
C     读入 MAT 文件的目录
      dir = matGetDir (mp, ndir)
      if (dir .eq. 0) then
         write (6, * ) 'Can''t read directory.'
         stop
      end if
C     将 dir 指向的数据拷贝到一个 FORTRAN 语言指针数组中
      call mxCopyPtrToInteger4 ( dir , adir , ndir )
```

```
C       从指针数组元素指向的内存中读取出字符串
        do 20 i=1, ndir
           call mxCopyPtrToCharacter (adir (i), names (i), 32)
   20 continue
C       输出 MAT 文件阵列的名字
        write (6, * ) 'Directory of Mat-file:'
        do 30 i=1, ndir
           write (6, * ) names (i)
   30 continue
C       使用函数 matClose 关闭 MAT 文件，并判断成功与否
        stat = matClose (mp)
        if (stat .ne. 0) then
           write (6, * ) 'Error closing "matdemo.mat".'
           stop
        end if
C       再次开启 MAT 文件
        mp = matOpen ('matdemo.mat', 'r')
        if (mp .eq. 0) then
           write (6, * ) 'Can't open "matdemo.mat".'
           stop
        end if
C       通过使用函数 matGetNextMatrix 依次从 MAT 文件中读取所有阵
C       列
        write (6, * ) 'Getting full array contents:'
        pa = matGetNextMatrix (mp)
        do while (pa .ne. 0)
C          获取阵列名字
           name = mxGetName (pa)
           i=mxGetM (pa)
           write (*, *) i
           write (6, * ) 'Retrieved ', name
           write (6, * ) ' With size ', mxGetM (pa), '-by-', mxGetN (pa)
           call mxFreeMatrix (pa)
           pa = matGetNextMatrix (mp)
        end do
C       关闭 MAT 文件
        stat = matClose (mp)
        if (stat .ne. 0) then
           write (6, * ) 'Error closing "matdemo.mat".'
           stop
        end if
        stop
C
```

```
    end
```

对该程序编译后执行，将显示如下内容：

```
E：\MATLAB\extern\examples\eng _ mat> matdemo2
Directory of Mat-file：
String
Numeric
Getting full array contents：
    1
Retrieved String
    With size 1-by- 33
    3
Retrieved Numeric
    With size 3-by- 3
```

4. matGetFull

功　能：从 MAT 文件中读取一个存储类型为满的阵列。

语　法：integer * 4 function matGetFull （mfp，name，m，n，pr，pi）
　　integer * 4 mfp，m，n，pr，pi
　　character * （ * ）name

说　明：函数 matGetFull 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。通过该函数，用户可以方便地从 MAT 文件中提取出一个存储类型为满的阵列，而且不需要使用 mxArray 结构体对象，可以直接得到有关阵列的信息，包括 m、n、pr 和 pi，这一点与函数 matGetMatrix 不同。函数 matGetFull 几个形式参数的含义分别如下：

- mfp 为一个指向 MAT 文件的指针，为一个输入参数；
- name 为希望从 MAT 文件中读取的阵列的名字，为一个输入参数；
- m 为从 MAT 文件中提取出阵列的行数，为一个输出参数；
- n 为从 MAT 文件中提取出阵列的列数，为一个输出参数；
- pr 为指向一片内存区域的指针，该内存区域通过使用函数 mxCalloc 进行分配，其中存放着从 MAT 文件中提取出阵列的实数部分的数据，在程序结束前，该内存区域必须使用函数 mxFree 进行释放，否则将导致内存泄漏；该参数为一个输出参数；
- pi 为指向一片内存区域的指针，该内存区域通过使用函数 mxCalloc 进行分配，其中存放着从 MAT 文件中提取出阵列的虚数部分的数据，在程序结束前，该内存区域必须使用函数 mxFree 进行释放，否则将导致内存泄漏；如果希望提取的阵列为实数阵列，那么 pi 为零；该参数为一个输出参数。

如果函数执行成功，函数将返回 0，否则将产生一个读文件错误。这里必须注意的一点是，该函数在后续的 MATLAB 版本中，将成为过时的函数，它

的功能完全可以通过函数 matGetMatrix，mxGetPr，mxGetPi，mxGetM 和 mxGetN 的组合使用来替代。

举　例：参见函数子程序 matPutFull 的范例程序。

5. matGetMatrix

功　能：从 MAT 文件中读取出一个阵列。

语　法：integer * 4 function matGetMatrix (mfp，name)

integer * 4 mfp

character * （ * ）name

说　明：函数 matGetMatrix 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。通过该函数，用户可以从 MAT 文件中获取一个阵列，如果函数成功执行，将返回一个指向新分配的 mxArray 结构体对象的指针，如果函数执行失败，将返回 0。必须注意，在程序结束前，必须对函数 matGetMatrix 分配的 mxArray 结构体进行删除，否则将导致内存泄漏。

举　例：参见函数子程序 matClose 的范例程序 matdemo1. f。

6. matGetNextMatrix

功　能：读取 MAT 文件中的下一个阵列。

语　法：integer * 4 function matGetNextMatrix (mfp)

integer * 4 mfp

说　明：函数 matGetNextMatrix 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。该函数的功能为读取输入参数 mfp 指向的处于开启状态的 MAT 文件中的下一个阵列的数据，并将返回一个新分配的 mxArray 结构体的指针。通过该函数，用户可以通过一个循环过程，依次访问 MAT 文件中所有的阵列。不过必须注意的一点是，对函数 matGetNextMatrix 的使用必须紧接在函数 matOpen 之后，也就是说在两者之间不能加入其他的 MAT 文件操作函数，否则函数 matGetNextMatrix 将不知道“下一个 (next)”的定义。如果到达文件结尾或者函数执行出错，函数将返回 0。在程序结束之前，必须将函数 matGetNextMatrix 分配的 mxArray 结构体删除，否则将导致内存泄漏。

举　例：参见函数子程序 matGetDir 的范例程序 matdemo2. f。

7. matGetString

功　能：从 MAT 文件中读取一个字符串类型的阵列。

语　法：integer * 4 function matGetString (mfp，name，str，strlen)

integer * 4 mfp，strlen

character * （ * ）name，str

说　明：函数 matGetString 为一个 FORTRAN 语言函数子程序，如果需要使用其返

回值，必须在程序中对函数加以类型说明。通过该函数，用户可以从 MAT 文件中提取指定名字的字符串类型的阵列，函数定义中的各形式参数的含义分别如下：

- mfp 为一个指向 MAT 文件的指针，为一个输入参数；
- name 为希望从 MAT 文件中读取的字符串类型的阵列的名字，为一个输入参数；
- str 为一个字符串变量，从 MAT 文件中读取的字符串类型的阵列的内容就存放在该变量中，为一个输出参数；
- strlen 为一个整型数量，代表了用户希望从字符串类型的阵列读取出字符的长度，一般将 strlen 的大小设置为阵列的元素个数，如果 strlen 小于阵列的元素个数，那么将只有前 strlen 个元素被读取；如果阵列由若干行组成，那么读取时将按列读取，并存放为一行；该参数为一个输入参数。

函数在不同的情况下有不同的返回值，代表了不同的含义，分别如下：

- 0 代表函数成功返回；
- 1 代表函数执行失败，因为希望读取的阵列不是字符串类型；
- 2 代表字符串阵列的元素个数大于 strlen；
- 3 代表函数执行出错。

举　例：参见函数子程序 matPutString 的范例程序。

8. matOpen

功　能：开启一个 MAT 文件。

语　法：integer * 4 function matOpen （filename，mode）

integer * 4 mfp

character * （ * ）filename，mode

说　明：函数 matOpen 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。通过该函数，用户可以开启一个 MAT 文件，如果函数执行成功，将返回一个指向 MAT 文件的指针，否则将返回 0。函数拥有两个输入参数，其中 filename 为一个字符串变量，字符串中包含了希望开启的 MAT 文件的文件名，mode 为一个字符变量，用来说明开启 MAT 文件的类型，它有以下几种取值：

- “r”，表示使用只读方式开启 MAT 文件；
- “u”，表示以可读和可写方式开启文件，即可以对文件中的某一些阵列进行更新；
- “w”，表示以只写方式开启文件，如果磁盘中存在同名的 MAT 文件，那么以这种方式开启 MAT 文件将删除原有文件中的所有内容；如果原来不存在同名的 MAT 文件，那么将创建一个新的文件；
- “w4”，表示创建一个 MATLAB 4.X 版本的 MAT 文件。

举　例：参见函数子程序 matClose 的范例程序 matdemo1.f。

9. matPutFull

功 能：向 MAT 文件中写入一个存储类型为满的矩阵。

语 法：integer * 4 function matPutFull (mfp, name, m, n, pr, pi)

integer * 4 mfp, m, n, pr, pi

character * (*) name

说 明：函数 matPutFull 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。通过该函数，用户可以方便地向 MAT 文件中写入一个存储类型为满的阵列，而且不需要使用 mxArray 结构体对象，而可以直接使用有关阵列的信息，包括 m、n、pr 和 pi，这一点与函数 matPutMatrix 不同。函数 matPutFull 几个形式参数的含义分别如下：

- mfp 为一个指向 MAT 文件的指针，为一个输入参数；
- name 为希望写入 MAT 文件的阵列的名字，为一个输入参数；
- m 为希望写入 MAT 文件的阵列的行数，为一个输入参数；
- n 为希望写入 MAT 文件的阵列的列数，为一个输入参数；
- pr 为指向一片内存区域的指针，该内存区域中存放着希望写入 MAT 文件的阵列的实数部分的数据；该参数为一个输入参数；
- pi 为指向一片内存区域的指针，该内存区域中存放着希望写入 MAT 文件的阵列的虚数部分的数据；该参数为一个输入参数。

如果函数执行成功，函数将返回 0，否则将产生一个写文件错误。如果在 MAT 文件中已经存在同名的阵列，那么函数将覆盖原来的阵列，如果不存在，将在 MAT 文件的尾部写入一个新的阵列。这里必须注意的一点是，该函数在后续的 MATLAB 版本中，将成为过时的函数，它的功能完全可以通过函数 matPutMatrix，mxSetPr，mxSetPi，mxSetM 和 mxSetN 的组合使用来替代。

举 例：下面是一个简单的范例程序，该程序从一个 MAT 文件中读取一个阵列，然后写入到另一个 MAT 文件中，程序的源代码如下：

```
      program main
C     程序中使用的函数和变量的声明
      integer matOpen, matClose, matPutFull, matGetFull
      integer mf1, mf2, stat
      integer m, n, pr, pi
      integer mfp
      double precision Areal (6)
      data Areal / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 /
C     创建一个名为 foo.mat 的 MAT 文件，并使用函数 matPutFull 写入一个阵
C     列，然后关闭该 MAT 文件
      mfp = matOpen ('foo.mat','w')
      stat = matPutFull (mfp,'A', 3, 2, Areal, 0)
```

```
      stat = matClose (mfp)
C     从 MAT 文件 foo.amt 中读取阵列，然后写入 MAT 文件 foo2.mat
      mf1 = matOpen ('foo.mat','r')
      mf2 = matOpen ('foo2.mat','w')
      stat = matGetFull (mf1,'A', m, n, pr, pi)
      stat = matPutFull (mf2,'A', m, n, pr, pi)
      stat = matClose (mf1)
      stat = matClose (mf2)
      stop
      end
```

10. matPutMatrix

功　能：向 MAT 文件写入一个 mxArray 结构体类型的对象。

语　法：integer * 4 function matPutMatrix (mfp, mp)
　　integer * 4 mp, mfp

说　明：函数 matPutMatrix 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。通过该函数，用户可以向 MAT 文件写入一个 mxArray 结构体类型的对象，如果函数成功执行，将返回 1，如果函数执行失败，将返回 0。必须注意，如果在 MAT 文件中已经存在同名的阵列，那么函数将覆盖原来的阵列，如果不存在，将在 MAT 文件的尾部写入一个新的阵列。新旧阵列的维数可以不一致。

举　例：参见函数子程序 matClose 的范例程序 matdemo1.f。

11. matPutString

功　能：向 MAT 文件写入一个字符串类型的阵列。

语　法：integer * 4 function matPutString (mfp, name, str)
　　integer * 4 mfp
　　character * (*) name, str

说　明：函数 matPutString 为一个 FORTRAN 语言函数子程序，如果需要使用其返回值，必须在程序中对函数加以类型说明。通过该函数，用户可以向 MAT 文件写入一个字符串类型的阵列。如果函数执行成功，函数将返回 0，否则为 1。函数的各输入参数的含义如下：

- mfp 为一个指向 MAT 文件的指针；
- name 为希望写入 MAT 文件的字符串类型的阵列的名字；
- str 为一个字符串变量，它包含了希望写入 MAT 文件的字符串类型的阵列的内容。

举　例：下面是一个简单的范例程序：

```
      program main
C     程序中使用的变量和函数的声明
      integer matOpen, matClose, matPutString, matGetString
```

```
      character * 100 str
      integer mfp1, stat, mfp2
C     向 MAT 文件 string.mat 写入一个字符串
      mfp1 = matOpen ('string.mat', 'w')
      stat = matPutString (mfp1,'A','Hello, world')
      stat = matClose (mfp1)
C     从 MAT 文件 string.mat 读取一个字符串
      mfp2 = matOpen ('string.mat', 'r')
      stat = matGetString (mfp2,'A', str, 10)
      stat = matClose (mfp2)
      write (6, * ) 'The String is ', str
      stop
      end
```

第 6 章　MATLAB 引擎函数库的使用

本章内容分为三节，首先分别针对 C 语言和 FORTRAN 语言，以 MATLAB 提供的示范程序为例，对 MATLAB 引擎的使用进行详细的介绍，然后对 MATLAB 引擎程序的建立进行说明，最后分类对 MATLAB 引擎函数库中的库函数进行全面的介绍。

6.1　MATLAB 引擎的使用

MathWorks 公司为了方便用户对 MATLAB 引擎的学习，提供了三个典型的范例程序，分别是基于 C 语言的 engdemo.c 和 engwindemo.c 以及基于 FORTRAN 语言的 fengdemo.f，它们均存放在目录

MATLAB 根目录\EXTERN\EXAMPLES\ENG_MAT

下。本节将分别以程序 engwindemo.c 和程序 fengdemo.f 为例，对基于 C 语言和 FORTRAN 语言的 MATLAB 引擎的使用进行讲解。

6.1.1　基于 C 语言的 MATLAB 引擎的使用

1. engwindemo.c 源代码

```
/* 程序段 (1) */
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "engine.h"
static double Areal [6] = { 1, 2, 3, 4, 5, 6 };
/* 程序段 (2) */
int PASCAL WinMain (HANDLE hInstance,
HANDLE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
{
    /* 程序段 (3) */
    Engine *ep;
    mxArray *T = NULL, *a = NULL, *d = NULL;
    char buffer [301];
    double *Dreal, *Dimag;
    double time [10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    /* 程序段 (4) */
    if (! (ep = engOpen (NULL)))
    {
```

```
      MessageBox ((HWND) NULL,
                  (LPSTR)"Can't start MATLAB engine",
                  (LPSTR) "Engwindemo.c", MB_OK);
      exit (-1);
}
/* 第一部分：向 MATLAB 发送数据，并在 MATLAB 中分析数据，
   对结果进行绘图 */
/* 程序段 (5) */
T = mxCreateDoubleMatrix (1, 10, mxREAL);
mxSetName (T, "T");
memcpy ( (char * ) mxGetPr (T), (char * ) time, 10 * sizeof (double));
/* 程序段 (6) */
engPutArray (ep, T);
/* 程序段 (7) */
engEvalString (ep, "D = .5. * (-9.8) .* T.^ 2;");
/* 程序段 (8) */
engEvalString (ep, "plot (T, D);");
engEvalString (ep, "title ('Position vs. Time for a falling object');");
engEvalString (ep, "xlabel ('Time (seconds)');");
engEvalString (ep, "ylabel ('Position (meters)');");
/* 第二部分：构造一个 mxArray 结构体，并将其传送给 MATLAB，
           然后计算该矩阵的特征值，并显示在 Windows 窗口中。 */
/* 程序段 (9) */
a = mxCreateDoubleMatrix (3, 2, mxREAL);
memcpy ( (char * ) mxGetPr (a), (char * ) Areal, 6 * sizeof (double));
mxSetName (a, "A");
/* 程序段 (10) */
engPutArray (ep, a);
/* 程序段 (11) */
engEvalString (ep, "d = eig (A * A')");
/* 程序段 (12) */
engOutputBuffer (ep, buffer, 300);
/* 程序段 (13) */
engEvalString (ep, "whos");
MessageBox ( (HWND) NULL, (LPSTR) buffer,
                          (LPSTR) "MATLAB - whos", MB_OK);
/* 程序段 (14) */
d = engGetArray (ep, "d");
/* 程序段 (15) */
engClose (ep);
/* 程序段 (16) */
if (d == NULL)
```

```
{
    MessageBox ( (HWND) NULL, (LPSTR)"Get Array Failed",
                 (LPSTR)"Engwindemo.c", MB_OK);
}
else
{
    Dreal = mxGetPr (d);
    Dimag = mxGetPi (d);
    if (Dimag)
      sprintf (buffer,"Eigenval 2: %g+%gi", Dreal [1], Dimag [1]);
else
      sprintf (buffer,"Eigenval 2: %g", Dreal [1]);
MessageBox ( (HWND) NULL, (LPSTR) buffer,
                 (LPSTR)"Engwindemo.c", MB_OK);
    mxDestroyArray (d);
}
/* 程序段 (17) */
mxDestroyArray (T);
mxDestroyArray (a);
return (0);
}
```

2. 程序说明

engwindemo.c 是一个基于 Windows 操作系统，使用C 语言开发的调用MATLAB引擎的 Windows SDK 应用程序。该程序从总体上可以分为两个部分，第一个部分构造了一个名为T 的 mxArray 类型结构体，并传送给 MATLAB，进行了一定的数据分析和绘图操作；第二部分则利用MATLAB 引擎完成了一个矩阵的特征计值计算任务，并将结果回传给客户应用程序，在 Windows 窗口中显示。下面按程序中的标号，对程序中的各段代码进行详细的解释，说明 MATLAB 引擎的使用：

程序段 (1) 为头文件和全局变量的定义，它对程序中所必需的一些头文件进行了包含，其中最为重要的是头文件 windows.h 和头文件 engine.h。头文件 windows.h 中包含了编写 Windows 应用程序所必需的一些宏、数据类型以及函数的定义，缺少了它，Windows 的应用程序根本无法运行；而 engine.h 中则包含了 MATLAB 引擎函数库中所有函数及相关数据类型的定义，同时对头文件 matrix.h 进行了包含，缺少了它们，将无法使用 MATLAB 引擎。此外，该程序段还定义了一个 double 类型的静态数组 Areal，其作用域和生存周期均为全局。

程序段 (2) 为主函数的定义，该语句定义了程序 engwindemo.c 的主函数 WinMain，该函数是程序运行时的入口，这与非 Windows 程序相比有较大不同，在非 Windows 程序中，主函数名为 main。主函数 WinMain 有四个参数，在编写 Windows 程序时，必须按格式写入，至于各个参数的含义，有兴趣的读者可以自行参考有关 Windows 编程方面的书籍，若仅仅想使用MATLAB 引擎，按上面的程序照抄即可。语句中的关键字 PAS-

CAL，声明了应用程序所使用的参数传送方式。

程序段（3）为变量的初始化，它对程序中需要的一些变量进行了定义，其中较为关键的是引擎指针ep的定义和mxArray结构体指针T、a、d的定义。这里必须注意一点，在一般情况下，最好将引擎变量和mxArray结构体变量都声明为指针类型，因为几乎所有的应用程序接口库中的库函数均以指针变量为参数，这样操作起来比较方便。如果一定要声明为普通变量，也可以，不过操作时需要使用取地址运算符&。

程序段（4）为MATLAB引擎的启动，这是非常关键的一个步骤，在使用MATLAB引擎之前，必须首先启动它，启动MATALB引擎可以使用引擎函数库中提供的函数engOpen，该函数若成功执行，将返回一个Engine类型的指针变量，在程序中，将这个变量赋给了指针ep；若函数执行失败，将返回一个空指针。有关该函数的具体使用方法，请参见6.3节。

程序段(5)为矩阵的构造和赋值，这段代码首先使用了两个以mx为前缀的函数mxCreateDoubleMatrix和mxSetName，创建了一个大小为1×10的双精度类型的mxArray结构体变量T，并将该结构体变量命名"T"，有关函数的详细使用说明请读者参见3.8节(有关mxArray结构体的定义请读者参阅前面的章节)；然后使用C语言的内存拷贝函数memcpy对所构造的结构体进行了赋值，这里必须非常注意函数参数中的指针类型的强制转换。

程序段(6)通过使用MATLAB引擎库的库函数engPutArray将程序段（5）中定义的变量T输送到MATLAB的工作空间，供后面的计算使用。

程序段(7)则通过MATLAB引擎库的库函数engEvalString向MATLAB发出一个计算指令，并且在MATLAB中计算完成。

程序段（8）则通过函数engEvalString调用了MATLAB中的一些内建的绘图函数，用于绘制程序段(7)中的计算结果。

至此，程序engwindemo.c的第一部分结束，下面为程序的第二部分，其中：

程序段(9)完成的功能与程序段(5)相同，它定义了另外一个大小为3×2的双精度类型的mxArray结构体变量a，并且对其进行了赋值，然后命名为"A"。

程序段(10)则将变量A输送到MATLAB的工作空间，供后续的计算使用；

程序段（11）使用了MATLAB引擎库的库函数engEvalString来调用MATLAB的内建数学函数eig，用于计算矩阵a的特征值。

程序段（12）通过MATLAB引擎库的库函数engOutputBuffer定义了一个大小为300的字符类型的缓冲内存区域，用于存放下一条MATLAB指令的输出。

程序段(13)通过函数engEvalString调用了MATLAB的内建函数whos，用于查看当前MATLAB工作空间中的各变量信息，这些输出的信息都将输出到缓存buffer中，并且使用Windows的API函数MessageBox将信息显示在一个对话框中。

程序段（14）用于从MATLAB环境中获得程序段（11）执行后输出的结果，类型为mxArray结构体。

程序段(15)与程序段(4)相对应，用于关闭MATLAB引擎，这里使用了MATLAB引擎库的库函数engClose。

程序段（16）为一个if语句，这段程序代码主要用来完成对程序段（14）的执行结

果进行判断，若程序段（14）执行不成功，将显示错误信息，报告程序员获取数据出现问题；若执行成功，则分别从该结构体中获取矩阵的实部和虚部，并且在 Windows 对话框中显示这些数据，然后对结构体进行了析构。

程序段（17）完成的工作是对前面所定义的结构体变量进行了析构，释放内存。

3. 运行结果

对 engwindemo. c 进行编译后，得到一个名为 engwindemo. exe 的可执行程序，直接在 Windows 下运行该程序，将依次显示以下的图形和对话框：

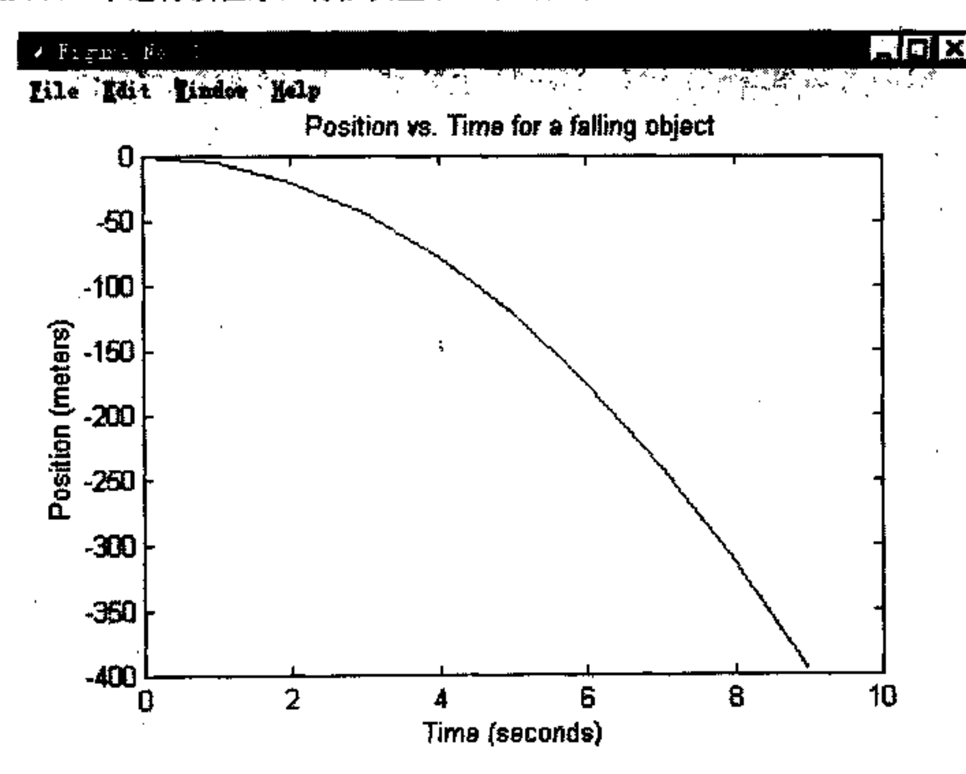


图 6. 1　程序 engwindemo 调用 MATLAB 绘制的图形

MATLAB - whos

```
Name    Size     Bytes  Class

A       3x2         48  double array
D       1x10        80  double array
T       1x10        80  double array
d       3x1         24  double array

Grand total is 29 elements using 232 bytes
```



图 6. 2　程序 engwindemo 显示的 MATLAB 的 whos 命令的输出



Eigenval 2: 0.597327



图 6. 3　程序 engwindemo 调用 MATLAB 计算特征值的输出结果

6.1.2 基于 FORTRAN 语言的 MATLAB 引擎的使用

1. fengdemo.f 源代码

```
C       fengdemo.f
        program main
C       程序段 (1)
        integer engOpen, engGetMatrix, mxCreateFull, mxGetPr
C       程序段 (2)
        integer ep, T, D, result
        double precision time (10), dist (10)
        integer stat, temp
        data time / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0 /
C       程序段 (3)
        ep = engOpen ('matlab ')
        if (ep .eq. 0) then
            write (6, * ) 'Can't start MATLAB engine'
            stop
        end if
C       程序段 (4)
        T = mxCreateFull (1, 10, 0)
        call mxSetName (T, 'T')
        call mxCopyReal8ToPtr (time, mxGetPr (T), 10)
C       程序段 (5)
        call engPutMatrix (ep, T)
C       程序段 (6)
        call engEvalString (ep, 'D = .5. * (-9.8) . * T. ^ 2;')
        call engEvalString (ep, 'plot (T, D);')
        call engEvalString (ep, 'title ("Position vs. Time")')
        call engEvalString (ep, 'xlabel ("Time (seconds)")')
        call engEvalString (ep, 'ylabel ("Position (meters)")')
C       程序段 (7)
        print * , 'Type 0 <return> to Exit'
        print * , 'Type 1 <return> to continue'
        read ( * , * ) temp
        if (temp.eq.0) then
            print * , 'EXIT!'
            stop
        end if
C       程序段 (8)
        call engEvalString (ep, 'close;')
C
```

```
      D = engGetMatrix (ep, 'D')
      call mxCopyPtrToReal8 (mxGetPr (D), dist, 10)
      print *, 'MATLAB computed the following distances:'
      print *, ' time (s) distance (m)'
      do 10 i=1, 10
         print 20, time (i), dist (i)
   20    format (' ', G10.3, G10.3)
   10 continue
C     程序段 (9)
      call mxFreeMatrix (T)
      call mxFreeMatrix (result)
      stat = engClose (ep)
C
      stop
      end
```

2. 程序说明

fengdemo.f 是 MATLAB 提供的一个基于 FORTRAN 语言使用 MATLAB 引擎函数库的范例程序，它的编写方法与普通的 FORTRAN 语言应用程序的编写方法完全一致，没有任何的特殊之处，惟一的不同是程序中使用了大量的 MATLAB 引擎函数和 MATLAB mx-函数，用于完成 MATLAB 引擎的开启，构建 mxArray 结构体对象，向 MATLAB 引擎传送数据，计算和关闭 MATLAB 引擎等任务。下面按程序中的编号，对程序中的各段代码进行详细的解释，说明 MATLAB 引擎的使用：

程序段 (1) 对程序中所使用到的 MATLAB 引擎库中的函数子程序和一些 mx-函数子程序进行了类型声明，这对于使用函数名进行值传递的 FORTRAN 语言函数子程序来说是非常关键的步骤。

程序段 (2) 对程序中所使用到的变量进行了类型声明，其中变量 ep 为引擎指针，T 和 D 为指向 mxArray 结构体对象的指针，time 和 dist 为双精度的浮点数组。

程序段 (3) 完成的主要功能是通过调用 MATLAB 引擎函数 engOpen，用于开启 MATLAB 引擎，并对函数执行的成功与否进行判断。

程序段 (4) 通过调用 MATLAB mx-函数子程序 mxCreateFull 构造了一个存储类型为满的 mxArray 结构体对象，并使用 mx-子例行程序 mxSetName 和 mxCopyReal8ToPtr 对该对象进行了命名和赋值。

程序段 (5) 完成的功能为使用 MATLAB 引擎子例行程序 engPutMatrix 将程序段 (4) 中创建的结构体对象传送到 MATLAB 的工作空间中。

程序段 (6) 通过多次调用 MATLAB 引擎子例行程序 engEvalString，完成了一定的计算和绘图任务。

程序段 (7) 为一段人机交互程序，当程序执行至此，将要求用户输入，以确定下一步程序的走向，若用户键入 0，则程序退出，键入则继续执行。

程序段 (8) 主要完成了三个任务，首先通过调用子例行程序 engEvalString 关闭前面生成的图形窗口，然后使用函数子程序 engGetMatrix 从 MATLAB 的工作空间中获取

程序段（6）的计算结果 D，最后调用了子例行程序 mxCopyPtrToReal8 从 mxArray 结构体对象 D 中获取数据并输出。

程序段（9）为一些后续的处理过程，包括释放 mxArray 结构体对象占用的内存和关闭 MATLAB 引擎。

程序中使用到的 MATLAB 引擎函数子程序和子例行程序将在本章的第三节中详细进行说明。

3. 执行结果

对程序 fengdemo. f 进行编译后，得到一个名为 fengdemo. exe 的可执行程序，执行该程序，将首先启动 MATLAB，在程序向其传送数据之后，MATLAB 将显示如下图形：

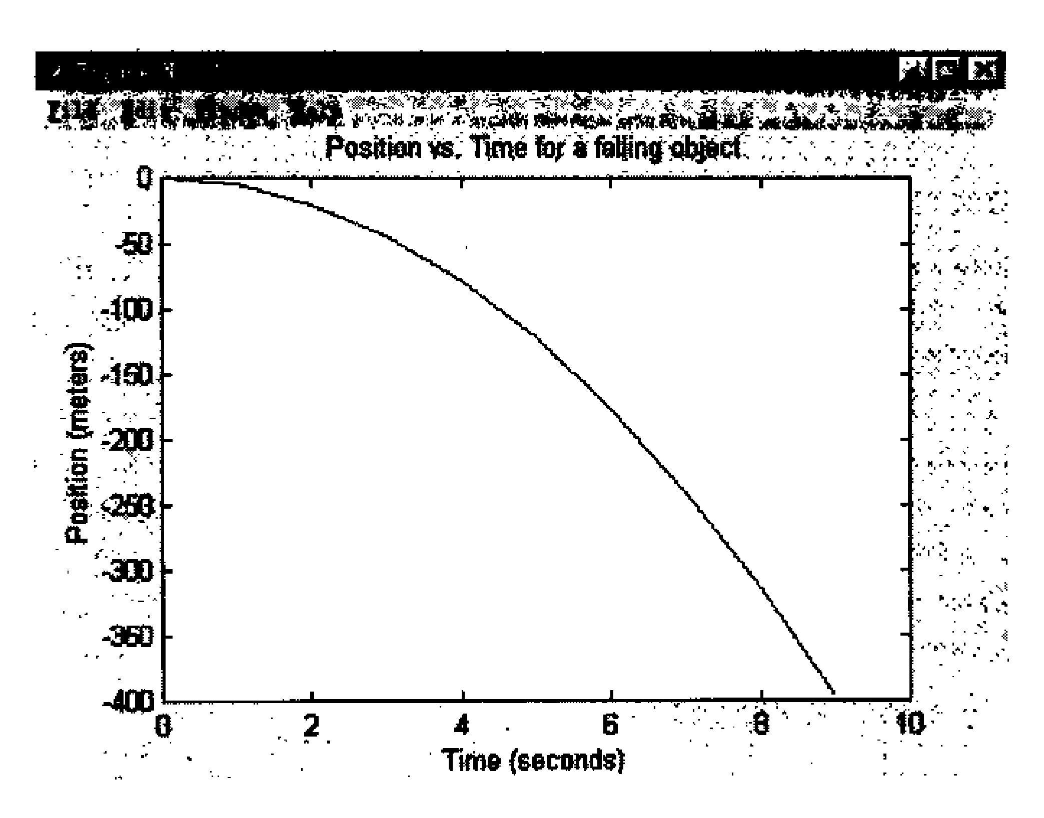


图 6.4　程序 fengdemo 调用 MATLAB 绘制的图形

然后程序将在 DOS 框中显示如下内容：

Type 0 <return> to Exit

Type 1 <return> to continue

当用户键入 1 后，程序将显示以下计算结果：

```
MATLAB computed the following distances:
time (s) distance (m)
1.00 -4.90
2.00 -19.6
3.00 -44.1
4.00 -78.4
5.00 -123.
```

```
6.00 -176.
7.00 -240.
8.00 -314.
9.00 -397.
10.0 -490.
```

最后程序将在释放内存，并关闭MATLAB引擎后终止运行。

6.2 MATLAB引擎程序的建立和调试

在本节中，我们将分别基于C语言和FOTRAN语言，对MATLAB引擎程序的建立和调试方法进行介绍，并给出直接在Microsoft VC++ 6.0集成环境和Microsoft Fortran PowerStation集成环境中建立和调试MATLAB引擎程序的方法。

6.2.1 C语言MATLAB引擎程序的建立和调试

1.C语言MATLAB引擎程序的建立

在Windows操作系统上，C语言MATLAB引擎程序的建立极为简单，用户在编写好源程序后，只需在MATLAB命令提示符下键入命令mex并辅以参数-f和相应的选项文件以及需要编译的源程序名即可，格式如下：

mex -f <matlab>\bin*optsfilename*.bat *filename*.c

其中*optsfilename*.bat为与用户系统中C语言编译器相对应的选项文件名，*filename*.c为用户编写的MATLAB引擎程序的源文件名，参数-f的含义为使用指定的选项文件对*filename*.c进行编译。关于选项文件的选择，读者可以参见本书的第2.4.1节。若引擎程序编译成功，mex命令将不返回任何信息，直接回到MATLAB命令提示符下，否则将给出一定的错误原因。

2.C语言MATLAB引擎程序的调试

C语言MATLAB引擎程序在编译链接通过后，并不意味着已经完全没有错误，这一点对于有经验的程序员来说是显而易见的。如果程序在运行时发生错误，最简单直接的查错方法就是调试（DEBUG），在一般的应用程序编制过程中，这是一个必不可少的步骤。下面我们以Microsoft VC++ 6.0为例，对C语言MATLAB引擎程序的调试进行讲解。

从大体上整个过程可以分为两步：首先开启集成编译环境，选择需要调试的MATLAB引擎程序的执行文件，将其调入到集成环境中，这时VC将为其创建一个工作空间；其次选择MATLAB引擎程序的源文件，将其调入到当前工程中。至此就完成了全部的设置工作，接下来就可以在程序中设置断点进行调试了。关于调试器的使用，请读者自行参见相关调试器的联机帮助。

这里必须非常注意一点，如果希望对一个C语言MATLAB引擎程序进行调试，那么在使用mex命令进行编译时，必须同时加入命令参数-g，格式如下：

mex -g -f <matlab>\bin*optsfilename*.bat *filename*.c

其中参数-g的含义为告诉编译器在编译链接程序时包含调试信息，否则程序将无法调

试。强烈建议用户在首次调试程序时使用-g 参数。

3. Microsoft VC++ 6.0 集成环境中 MATLAB 引擎程序的建立和调试

习惯了使用各种集成环境的读者，可能会觉得 MATLAB 提供的这种程序开发方法非常的不方便。建立一个 MATLAB 引擎程序，首先必须在某个编辑器中进行源代码的编写，然后存盘后回到 MATLAB 的工作环境中进行编译，若发现错误，则必须回到原来的文件编辑器，按照 MATALB 的错误提示，逐行地对源代码进行查错修改，之后再回到 MATLAB 的工作环境中进行编译，如此往复，直至程序没有错误为止，一个 MATLAB 引擎程序才宣告完成。整个过程需要来回地在文件编辑器和 MATLAB 工作环境之间切换，非常不方便，而且过程中还没有加入调试步骤，否则将更加麻烦。

为了方便读者，我们将给出一种在 Microsoft VC++ 6.0 集成环境中的建立和调试 MATLAB 引擎程序的方法，其基本步骤如下：

第一步，启动 Microsoft VC++ 6.0 集成环境，选择 File 下拉式菜单中的 New 选项，这时将弹出如图 6.5 所示的对话框，用户可以选择其中三种类型的应用程序创建工程，分别为 MFC AppWizard (exe)，Win32 Application 和 Win32 console Application，为了简便起见，我们以 Win32 console Application 为例进行说明。选择 Win32 console Application 项目，并在 Project name 编辑框中输入项目名，并点击 OK 按钮；

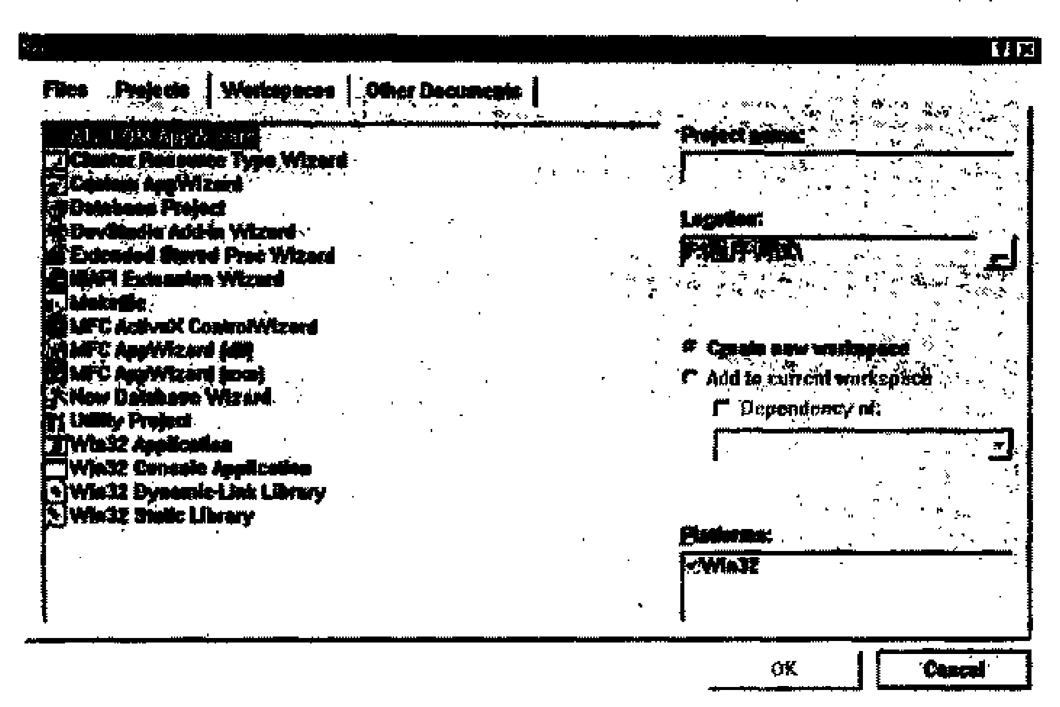


图 6.5　VC++ 6.0 File New 对话框

第二步，完成以上操作后，VC++ 6.0 编译器将弹出如图 6.6 所示的对话框，提示用户选择创建 Win32 console Application 程序的类型，这里我们选择其中的最为简单的类型 An Empty project，然后选择 Finish 按钮，创建该项目。

第三步，在项目工程创建完毕之后，选择下拉式菜单 Tools 中的菜单项 Options，将弹出 Options 对话框，选择其中的 Directories 属性页，如图 6.7 所示，在其中的 Show directories for 下拉式选项框中分别选择 Include Files 和 Library Files，在下部的编辑框中

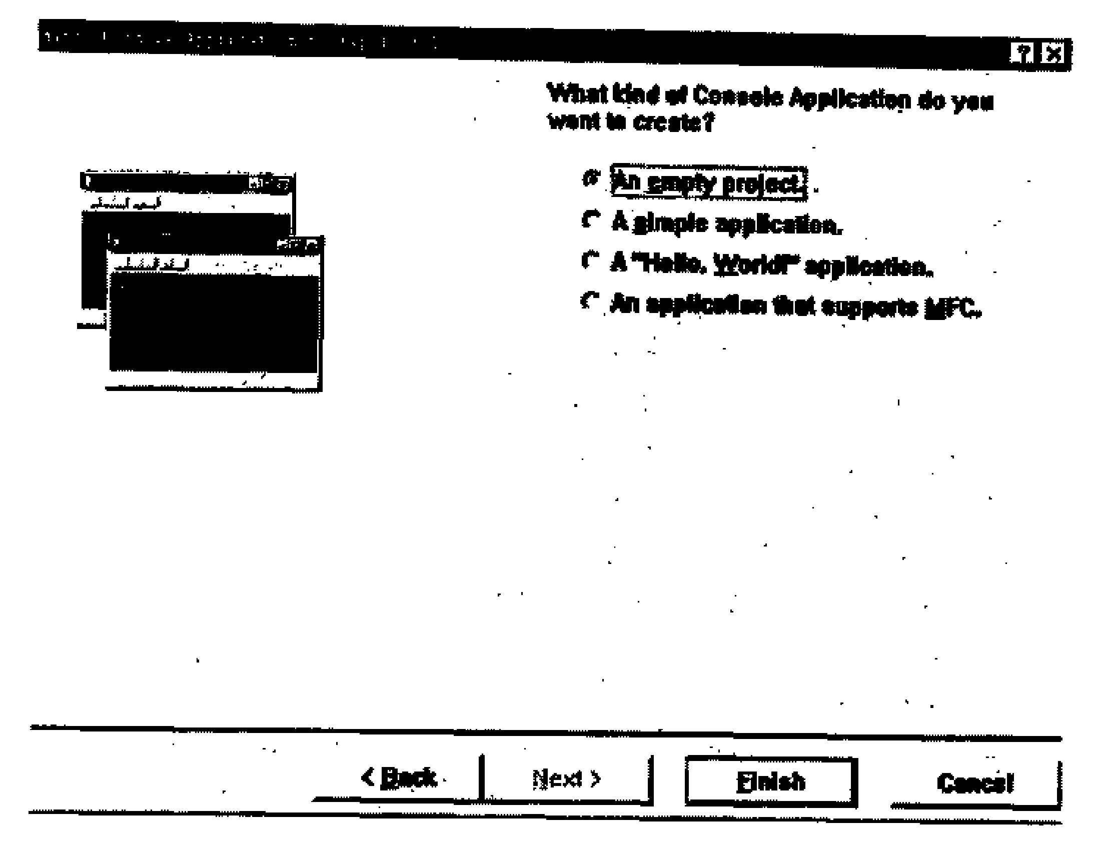


图 6.6　Win32 console Application 程序的类型选择对话框

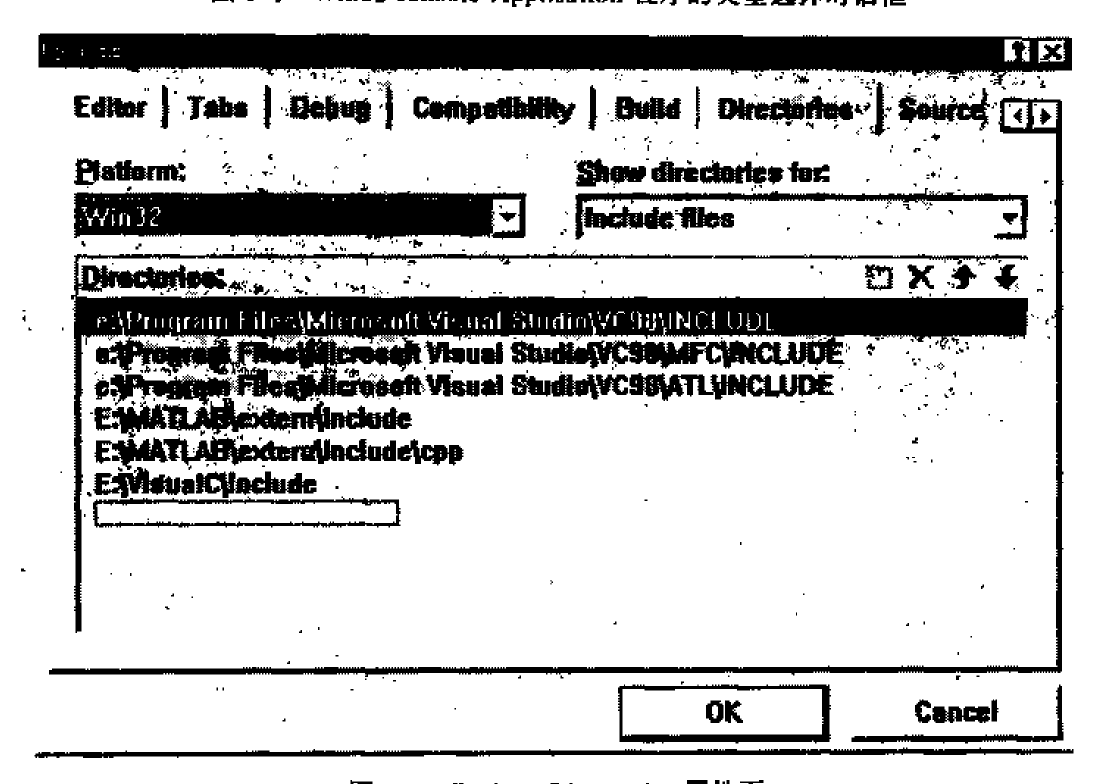


图 6.7　Options Directories 属性页

输入以下路径：

MATLAB 根目录\EXTERN\INCLUDE

MATLAB 根目录\EXTERN\LIB

然后选择 OK 按钮；

第四步，在DOS命令框状态下，进入用户安装Microsoft VC++ 6.0的目录，如d:\Pogram files\Microsoft Visual Studio\VC98，并且进入该目录下的子目录\bin，按下面的格式运行该目录下的命令lib：

lib /def：%MATLAB%\extern\include\libmx.def /machine：ix86 /OUT：LIB_NAME1.lib /NOLOGO

lib /def：%MATLAB%\extern\include\libeng.def /machine：ix86 /OUT：LIB_NAME2.lib /NOLOGO

lib /def：%MATLAB%\extern\include\libmat.def /machine：ix86 /OUT：LIB_NAME3.lib /NOLOGO

执行完这三条命令后，用户可以得到三个静态链接库文件，分别为LIB_NAME1.lib、LIB_NAME2.lib和LIB_NAME3.lib。命令中%MATLAB%代表本机上安装MATLAB的根目录，在执行这些命令的过程中，必须加以替换，如D：\MATLAB；

这里可以明确一点，一旦三个静态链接库文件生成后，就可以反复使用，而无须对每一个项目进行重新建立；

第五步，选择下拉式菜单Project中的菜单项Add To Project>>Files将第四步中生成的三个lib库文件添加到当前项目中，同时将用户编写的MATLAB引擎程序的源文件也添加到当前项目中。

当完成了以上五步工作之后，用户就可以在VC++中对MATLAB引擎程序进行编译和调试了。对于MFC AppWizard（exe）和Win32 Application类型的MATLAB引擎程序，步骤大致相同。

6.2.2 FORTRAN语言MATLAB引擎程序的建立和调试

1.FORTRAN语言MATLAB引擎程序的建立

在Windows操作系统上，FORTRAN语言MATLAB引擎程序的建立与C语言MATLAB引擎程序的建立过程极为相似。用户在编写好源程序后，只需在MATLAB命令提示符下键入命令mex并辅以参数-f和相应的选项文件以及需要编译的源程序名即可，格式如下：

mex -f <matlab>\bin*optsfilename*.bat *filename*.f

惟一的不同就是命令中*optsfilename*.bat为与用户系统中FORTRAN语言编译器相对应的选项文件名，*filename*.f为用户编写的MATLAB引擎程序的源文件名，参数-f的含义为使用指定的选项文件对*filename*.f进行编译。关于选项文件的选择，读者可以参见本书的第2.4.1节。若引擎程序编译成功，mex命令将不返回任何信息，直接回到MATLAB命令提示符下，否则将给出一定的错误原因。

2.FORTRAN语言MATLAB引擎程序的调试

与C语言MATLAB引擎程序类似，FORTRAN语言MATLAB引擎程序在编译链接通过后，并不意味着已经完全没有错误，同样需要一个调试过程。下面我们以Microsoft Fortran PowerStation为例，对FORTRAN语言MATLAB引擎程序的调试进行说明。

整个调试过程具体可以分为两步：首先进入Microsoft Fortran PowerStation集成环

境，选择下拉式菜单 File 中的菜单项 Open Workspace，这时将弹出一个文件选择对话框，用户从中选择希望调试的 MATLAB 引擎程序的可执行文件，将其调入当前工作空间；第二步，选择下拉式菜单 File 中的菜单项 Open 将 MATLAB 引擎程序的源程序调入工作空间。至此就完成了全部的设置工作，接下来就可以在程序中设置断点进行调试了。关于调试器的使用，请读者自行参见相关调试器的联机帮助。

这里同样必须非常注意一点，如果希望对一个 FORTRAN 语言 MATLAB 引擎程序进行调试，那么在使用 mex 命令进行编译时，必须同时加入命令参数-g，格式如下：

mex -g -f <matlab>\bin*optsfilename*. bat *filename*. c

其中参数-g 的含义为告诉编译器在编译链接程序时包含调试信息，否则程序将无法调试。强烈建议用户在首次调试程序时使用-g 参数。

3. Microsoft Fortran PowerStation 集成环境 MATLAB 引擎程序的建立和调试

出于同样的目的，即为了简化整个 FORTRAN 语言 MATLAB 引擎程序的建立和调试过程，我们将给出一种在 Microsoft Fortran PowerStation 集成环境中完成整个程序编译和调试过程的方法。其具体步骤如下：

第一步，进入 Microsoft Fortran PowerStation 集成环境，从下拉式菜单 File 选取 New 菜单项，系统将弹出一个如图 6.8 所示的选项框，选择其中的“Project Workspace”项，并点击 OK 按钮，用于创建一个新的项目空间；

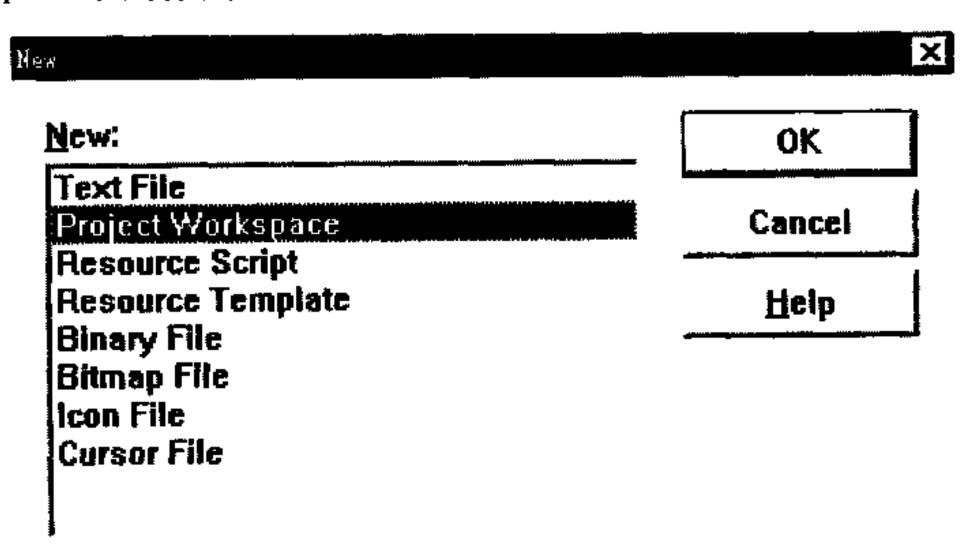


图 6.8　New 选项框

第二步，在关闭 New 选项框之后，系统将弹出 New Project Workspace 选项框，如图 6.9 所示，用户可以选择其中多种类型的项目用于创建 MATLAB 引擎程序，例如可以为 Application 或 Console Application，也可以为 QuickWin Application 或 Standard Graphics Application，为了说明方便，这里我们使用 Console Application 类型。点取 Type 选项框中的 Console Application 项，并在 Name 编辑框中输入项目名字后，单击 Create 按钮，创建项目；

第三步，在项目工程创建完毕之后，选择下拉式菜单 Tools 中的菜单项 Options，将

图 6.9　New Project Workspace 选项框

图 6.10　Options 对话框

弹出 Options 对话框，选择其中的 Directories 属性页，如图 6.10 所示，在其中的 Show directories for 下拉式选项框中分别选择 Include Files 和 Library Files，在下部的编辑框中输入以下路径：

MATLAB 根目录\EXTERN\INCLUDE

MATLAB 根目录\EXTERN\LIB

然后选择 OK 按钮；

第四步，在 DOS 命令框状态下，进入用户安装 Microsoft FORTRAN PowerStation 的目录，如 d:\msdev，并且进入该目录下的子目录\bin，按下面的格式运行该目录下的命令 lib：

lib /def:%MATLAB%\extern\include\dfmx.def /machine:ix86 /OUT: LIB _ NAME1.lib /

NOLOGO

lib /def:%MATLAB%\extern\include\dfmat.def /machine:ix86 /OUT: LIB _ NAME2.lib /NOLOGO

lib /def:%MATLAB%\extern\include\dfeng.def /machine:ix86 /OUT: LIB _ NAME3.lib /NOLOGO

以产生三个分别名为 LIB _NAME1.lib，LIB _NAME2.lib 和 LIB _NAME3.lib 的静态链接库文件，其中%MATLAB%表示用户安装 MATLAB 的根目录，如 d:\matlab，一旦库文件成功生成后，可以应用在所有的 MATLAB 引擎程序的工程之中，而无须重复生成；

第五步，选择下拉式菜单 Insert 的菜单项 Files into Project，选择库文件第四步中生成的三个 lib 文件，将它们嵌入到当前的工程中；同时选择用户的 MATLAB 引擎程序的源文件，将其也嵌入到当前的工程中。

完成了以上五步工作之后，用户就可以在 Microsoft Fortran PowerStation 集成环境中对 MATLAB 引擎程序进行编译和调试了。对于其他项目类型的 MATLAB 引擎程序，步骤大致相同。

6.3 MATLAB 引擎函数

本节将分别基于 C 语言和 FORTRAN 语言，对 MATLAB 引擎函数库中的函数使用进行说明。

6.3.1 C 语言引擎函数的使用说明

在 MATLAB 引擎函数库中，总共提供了 13 个 C 语言的引擎函数，它们的声明分别如下：

int engClose (Engine *ep)

int engEvalString (Engine *ep, const char *string)

mxArray *engGetArray (Engine *ep, const char *name)

engGetFull (*Obsolete*)

engGetMatrix (*Obsolete*)

Engine *engOpen (const char *startcmd)

int engOutputBuffer (Engine *ep, char *p, int n)

int engPutArray (Engine *ep, const mxArray *mp)

engPutFull (*Obsolete*)

engPutMatrix (*Obsolete*)

engSetEvalCallback (*Obsolete*)

engSetEvalTimeout (*Obsolete*)

engWinInit (*Obsolete*)

所有这些函数均在头文件 engine.h 中予以声明，在使用它们时，必须对该头文件进行包含。下面我们对这些函数进行逐一说明。

1. engClose

功　能：关闭 MATLAB 引擎。

语　法：#include "engine. h"

int engClose(Engine *ep);

说　明：通过该函数用户可以关闭一个已经处于开启状态的 MATLAB 引擎。该函数在执行时，首先向 MATLAB 引擎发出一个关闭命令，然后断开与 MATLAB 引擎的连结。若函数执行成功将返回 0，否则返回 1。一般来说，导致函数执行失败的主要原因是试图再次关闭一个已经关闭的 MATLAB 引擎。该函数拥有一个类型为 Engine 指针的输入参数。

举　例：参见 6. 1. 1 节程序 engwindemo. c。

2. engEvalString

功　能：执行一个用字符串表示的 MATLAB 表达式。

语　法：#include "engine. h"

int engEvalString (Engine *ep, const char *string);

说　明：函数 engEvalString 包含两个输入参数，其中第一个参数 ep 为一个 Engine 类型的指针，第二个参数 string 为一个字符指针，代表了一个字符串，该字符串应该为一个合法的 MATLAB 表达式。若该函数执行成功将返回 0，否则返回 1，一般在下列两种情况下，函数将会执行失败：第一，MATLAB 引擎已经关闭；第二，参数 string 所包含的字符串为非法的 MATLAB 表达式。在不同的操作系统上，函数与 MATLAB 的通信方式也不相同，例如在 UNIX 系统上，函数通过管道（pipe）连结到 MATLAB 的 stdin 上，向 MATLAB 输入命令；而在 Windows 系统上，函数与 MATLAB 的通行则是通过 ActiveX 来完成的。

举　例：参见 6. 1. 1 节程序 engwindemo. c。

3. engGetArray

功　能：从 MATLAB 的工作空间中获取一个变量。

语　法：#include "engine. h"

mxArray *engGetArray (Engine *ep, const char *name);

说　明：在使用函数 engGetArray 时必须提供两个输入参数，一个为指向 MATLAB 引擎的指针变量 ep，另一个为指向某个字符串的字符指针 name。通过这个函数，用户可以从 MATLAB 的工作空间中获取使用参数 name 指定的 mxAray 结构体的内容，同时复制给一个新分配 mxArray 结构体，并将该结构体返回给用户。若该函数执行成功，将得到一个 mxArray 结构体对象的指针，否则返回值为 NULL。一般情况下，导致函数执行失败的原因主要有两种：第一为在 MATLAB 工作空间中不存在指定名字的变量；第二为在 V4 模式下使用了 MATLAB 4. X 版本所不支持的数据类型。

举　例：参见 6.1.1 节程序 engwindemo.c。

4. engGetFull (*Obsolete*)

说　明：函数 engGetFull 为一个过时的函数，在 MATLAB 5.X 版本的引擎程序中不应被继续使用，其功能可以通过组合使用引擎函数 engGetArray 和 mx-函数 mxGetM、mxGetN、mxGetPr、mxGetPi 来完成，用户可以使用它们在自己的引擎程序中自定义一个同名函数，用于完成同样的功能，源代码如下：

```
int engGetFull (
    Engine *ep, /* 引擎指针 */
    char *name, /* 满 mxArray 结构体对象名 */
    int *m, /* 返回的阵列行数 */
    int *n, /* 返回的阵列的列数 */
    double **pr, /* 返回的阵列实数部分的指针 */
    double **pi /* 返回的阵列虚数部分的指针 */
    )
    {
    mxArray *pmat;
    /* 获取阵列的副本 */
    pmat = engGetArray (ep, name);
    /* 判断函数 engGetArray 是否执行成功 */
    if (! pmat)
    return (1);
    if (! mxIsDouble (pmat))
    {
        mxDestroyArray (pmat);
        return (1);
    }
    /* 获取各返回值 */
    *m = mxGetM (pmat);
    *n = mxGetN (pmat);
    *pr = mxGetPr (pmat);
    *pi = mxGetPi (pmat);
    /* 置空实数和虚数部分的指针，以便于释放 */
    mxSetPr (pmat, NULL);
    mxSetPi (pmat, NULL);
    mxDestroyArray (pmat);
    return (0);
}
```

为了与已有的程序代码保持兼容，用户可以在使用 mex 命令对这些程序进行编译时使用-V4 命令参数。

5. engGetMatrix（*Obsolete*）

说　明：函数 engGetMatrix 为一个过时的引擎函数，在 MATLAB 5. X 版本的引擎程序中不应被继续使用，其功能可以通过新的引擎函数 engGetArray 所取代。为了与已有的程序代码保持兼容，用户可以在使用 mex 命令对这些程序进行编译时使用-V4 命令参数。

6. engOpen

功　能：启动 MATLAB 引擎。

语　法：

```
#include "engine.h"
Engine *engOpen（const char *startcmd）;
```

说　明：通过函数 engOpen，用户可以在自己的应用程序中，在后台启动一个 MATLAB 进程，用于完成一定的计算任务，其返回值为一个 Engine 类型的指针变量，若函数执行成功将返回开启的 MATLAB 引擎的指针，否则为 NULL，其输入参数为一个字符指针 startcmd，函数通过使用该输入参数指向的字符串所包含的命令与 MATLAB 建立一个连结，这里值得注意的是在 Windows 操作系统上，该字符指针必须为 NULL。在 UNIX 操作系统上，如果输入参数 startcmd 为 NULL 或者为空字符串，则函数直接使用命令 matlab 启动本机上的 MATLAB；如果输入参数 startcmd 为一个主机名，函数则将该主机名嵌入到一个更大的字符串

```
"rsh hostname \"/bin/csh -c 'setenv DISPLAY\
            hostname: 0; matlab'\""
```

中，用于启动指定主机上的 MATLAB；如果输入参数 startcmd 为其他一些字符串，如包含了空格或者其他一些非字符非数字的符号，这时函数将对输入参数 startcmd 不予理睬，直接启动 MATLAB。在 UINX 系统上函数 engOpen 的执行过程可以分为如下步骤：

- 创建两个管道；
- 派生一个进程，并且通过两个管道建立 MATLAB 和引擎程序间的通信；
- 执行命令，运行 MATLAB。

在 Windows 操作系统上，相对简单许多，函数开启一个 ActiveX 通道与 MATLAB 进行通信。

举　例：参见 6.1.1 节程序 engwindemo.c。

7. engOutputBuffer

功　能：确定存放 MATLAB 输出结果的缓冲区域。

语　法：#include "engine.h"

int engOutputBuffer (Engine *ep, char *p, int n);

说　明：通过引擎函数 engOutputBuffer，用户可以为由引擎指针 ep 所指向的引擎设置一个输出缓冲区，将 MATLAB 输出到屏幕上的内容保存在其中，其长度可以由参数 n 确定，位置由字符指针 p 来确定。在默认情况下，通过引擎函数 engEvalString 所执行的所有的 MATLAB 操作所产生的屏幕输出均被抛弃，而函数 engOutputBuffer 则告诉所有的后继 engEvalString 操作，将屏幕输出保留到缓冲 p 中。

举　例：参见 6.1.1 节程序 engwindemo.c。

8. engPutArray

功　能：将 mxArray 结构体类型变量输送到 MATLAB 的工作空间中。

语　法：#include "engine.h"

int engPutArray (Engine *ep, const mxArray *mp);

说　明：引擎函数 engPutArray 包含两个输入参数，分别为引擎指针 ep 和 mxArray 结构体对象指针。执行该函数，允许用户将一个 mxArray 结构体类型的变量输送到引擎 ep 中，如果在当前程序的工作空间中不存在所指定的 mxArray 结构体，函数会自动创建。如果函数执行成功，返回值为 0，否则为 1。

举　例：参见 6.1.1 节程序 engwindemo.c。

9. engPutFull (*Obsolete*)

说　明：函数 engPutFull 为一个过时的函数，在 MATLAB 5.X 版本的引擎程序中不应被继续使用，其功能可以通过组合使用引擎函数 engPutArray 和 mx-函数 mxCreateDoubleMatrix 来完成，用户可以使用它们在自己的引擎程序中自定义一个同名函数，用于完成同样的功能，源代码如下：

```
int engPutFull (
    Engine *ep, /* 引擎指针 */
    char *name, /* 阵列名字 */
    int m, /* 阵列行数 */
    int n, /* 阵列列数 */
    double *pr, /* 实部指针 */
    double *pi /* 虚部指针 */
)
{
    mxArray *pmat;
    int retval;
    /* 构造一个双精度类型的复数矩阵 */
    pmat = mxCreateDoubleMatrix (0, 0, mxCOMPLEX);
    mxSetName (pmat, name);
    mxSetM (pmat, m);
```

```
        mxSetN (pmat, n);
        mxSetPr (pmat, pr);
        mxSetPi (pmat, pi);
        /* 向 MATLAB 输出 */
        retval = engPutArray (ep, pmat);
        /* 置空实数和虚数部分的指针，以便于释放 */
        mxSetPr (pmat, NULL);
        mxSetPi (pmat, NULL);
        mxDestroyArray (pmat);
        return (retval);
    }
```

为了与已有的程序代码保持兼容，用户可以在使用 mex 命令对这些程序进行编译时使用－V4 命令参数。

10. engPutMatrix (*Obsolete*)

说　明：函数 engPutMatrix 为一个过时的引擎函数，在 MATLAB 5. X 版本的引擎程序中不应被继续使用，其功能可以通过新的引擎函数 engPutArray 所取代。为了与已有的程序代码保持兼容，用户可以在使用 mex 命令对这些程序进行编译时使用－V4 命令参数。

11. engSetEvalCallback (*Obsolete*)

说　明：函数 engSetEvalCallback 为一个过时的引擎函数，在 MATLAB 5. X 版本的引擎程序中不应被继续使用。

12. engSetEvalTimeout (*Obsolete*)

说　明：函数 engSetEvalTimeout 为一个过时的引擎函数，在 MATLAB 5. X 版本的引擎程序中不应被继续使用。

13. engWinInit (*Obsolete*)

说　明：函数 engWinInit 为一个过时的引擎函数，在 MATLAB 5. X 版本的引擎程序中不应被继续使用。

6.3.2　FORTRAN 语言引擎函数的使用说明

在 MATLAB 引擎函数库中，总共提供了 8 个 FORTRAN 语言的引擎函数，它们的声明分别如下：

```
integer * 4 function engClose (ep)
integer * 4 function engEvalString (ep, command)
integer * 4 function engGetFull (ep, name, m, n, pr, pi)
integer * 4 function engGetMatrix (ep, name)
integer * 4 function engOpen (startcmd)
```

```
integer * 4 function engOutputBuffer (ep, p)
integer * 4 function engPutFull (ep, name, m, n, pr, pi)
integer * 4 function engPutMatrix (ep, mp)
```

所有这些函数均被声明为函数子程序，如果希望使用它们的返回值，那么在使用前必须进行类型声明。下面我们对这些函数进行逐一说明。

1. engClose

功　能：关闭 MATLAB 引擎。

语　法：integer * 4 function engClose (ep)
integer * 4 ep

说　明：通过该函数，用户可以关闭一个已经处于开启状态的 MATLAB 引擎。该函数在执行时，首先向 MATLAB 引擎发出一个关闭命令，然后断开与 MATLAB 引擎的连结。若函数执行成功将返回 0，否则返回 1。一般来说，导致函数执行失败的主要原因是试图再次关闭一个已经关闭的 MATLAB 引擎。该函数拥有一个类型为 Engine 指针的输入参数。

举　例：参见 6.1.2 节程序 fengdemo. f。

2. engEvalString

功　能：执行一个用字符串表示的 MATLAB 表达式。

语　法：integer * 4 function engEvalString (ep, command)
integer * 4 ep
character * (*) command

说　明：函数 engEvalString 包含两个输入参数，其中第一个参数 ep 为一个 Engine 类型的指针，第二个参数 string 为一个字符指针，代表了一个字符串，该字符串应该为一个合法的 MATLAB 表达式。若该函数执行成功将返回 0，否则返回 1，一般在下列两种情况下，函数将会执行失败：第一，MATLAB 引擎已经关闭；第二，参数 string 所包含的字符串为非法的 MATLAB 表达式。在不同的操作系统上、函数与 MATLAB 的通信方式也不相同，例如在 UNIX 系统上，函数通过管道 (pipe) 连结到 MATLAB 的 stdin 上，向 MATLAB 输入命令；而在 Windows 系统上，函数与 MATLAB 的通行则是通过 ActiveX 来完成的。

举　例：参见 6.1.2 节程序 fengdemo. f。

3. engGetFull

功　能：从引擎中读取一个满存储类型的 mxArray 结构体。

语　法：integer * 4 function engGetFull (ep, name, m, n, pr, pi)
integer * 4 ep, m, n, pr, pi
character * (*) name

说　明：函数 engGetFull 拥有六个输入参数，含义分别如下：

- ep 为引擎指针
- name 为用户希望读取的 mxArray 结构体的名字
- m 为返回的指定的 mxArray 结构体的行数
- n 为返回的指定的 mxArray 结构体的列数
- pr 为返回的指定的 mxArray 结构体的实数部分的指针
- pi 为返回的指定的 mxArray 结构体的虚数部分的指针

如果该函数执行成功，返回值为 0，否则为 1。函数在执行过程中，通过使用 mx-函数 mxCalloc 为阵列的实数和虚数部分分配内存，用户在使用完毕后，必须使用 mx-函数 mxFree 对这些内存进行释放。此外还必须明确一点，函数 engGetFull 只能对存储类型为满的阵列进行操作。

该函数在后续的版本中，将被列为过时的函数，希望用户尽可能地不要使用该函数，可以通过组合使用引擎函数 engGetMatrix 和 mx-函数 mxGetM、mxGetN、mxGetPr、mxGetPi 来完成该函数的功能。

4. engGetMatrix

功　能：从 MATLAB 引擎的工作空间中获取 mxArray 结构体类型变量。

语　法：integer * 4 function engGetMatrix (ep, name)

integer * 4 ep

character * (*) name

说　明：通过引擎函数 engGetMatrix，用户可以从 MATLAB 引擎的工作空间中获取 mxArray 结构体类型变量，输入参数 ep 用于指定 MATLAB 引擎，为一个指针变量，name 用于确定希望读取的 mxArray 结构体的名字，为字符串变量。函数执行成功，将返回一个指向新分配的 mxAray 结构体对象，否则返回 0。这里必须非常注意一点，在退出引擎程序前必须对新分配的 mxAray 结构体对象进行释放操作。

举　例：参见 6.1.2 节程序 fengdemo.f。

5. engOpen

功　能：开启一个 MATLAB 引擎。

语　法：integer * 4 function engOpen (startcmd)

integer * 4 ep

character * (*) startcmd

说　明：通过函数 engOpen，用户可以在自己的应用程序中，在后台启动一个 MATLAB 进程，用于完成一定的计算任务，其返回值为一个 Engine 类型的指针变量，若函数执行成功将返回开启的 MATLAB 引擎的指针，否则为 0，其输入参数为一个字符串变量 startcmd，函数通过使用该字符串所包含的命令与 MATLAB 建立一个连结。在 UNIX 操作系统上，如果输入参数 startcmd 为空字符串，则函数直接使用命令 matlab 启动本机上的 MATLAB；如果输入参数 startcmd 为一个主机名，函数则将该主机名迁入到一个

更大的字符串

"rsh hostname \"/bin/csh -c 'setenv DISPLAY\
hostname：0；matlab'\""

中，用于启动指定主机上的MATLAB；如果输入参数startcmd 为其他一些字符串，如包含了空格或者其他一些非字符非数字的符号，这时函数将对输入参数 startcmd 不予理睬，直接启动 MATLAB。在 UINX 系统上函数 engOpen 的执行过程可以分为如下步骤：

- 创建两个管道；
- 派生一个进程，并且通过两个管道建立 MATLAB 和引擎程序间的通信；
- 执行命令，运行 MATLAB；

在 Windows 操作系统上，相对简单许多，函数开启一个 ActiveX 通道与 MATLAB 进行通信。

举　例：参见 6.1.2 节程序 fengdemo.f。

6. engOutputBuffer

功　能：指定 MATLAB 的屏幕输出缓冲。

语　法：integer * 4 function engOutputBuffer (ep，p)
integer * 4 ep
character * n p

说　明：通过引擎函数 engOutputBuffer，用户可以为由引擎指针 ep 所指向的引擎设置一个输出缓冲区，将 MATLAB 输出到屏幕上的内容保存到其中，该缓冲区通过字符串变量 p 来指定。在默认情况下，通过引擎函数 engEvalString 所执行的所有的 MATLAB 操作所产生的屏幕输出均被抛弃，而函数 engOutputBuffer 则告诉所有的后继 engEvalString 操作，将屏幕输出保留到缓冲 p 中。

举　例：程序代码如下：

```
      program main
C     引擎函数类型声明及指针声明
      integer engOpen, engGetMatrix, mxCreateFull, mxGetPr
      integer ep, T, D, result
C     相关变量声明
      double precision time (10), dist (10)
      integer stat, temp
      data time / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0 /
      character * 1000 p
C     引擎开启，并判断成功与否
      ep = engOpen ('matlab ')
      if (ep .eq. 0) then
```

```
          write (6, * ) 'Can't start MATLAB engine'
          stop
       endif
C      创建满矩阵并输出到 MATLAB 引擎
       T = mxCreateFull (1, 10, 0)
       call mxSetName (T, 'T')
       call mxCopyReal8ToPtr (time, mxGetPr (T), 10)
       call engPutMatrix (ep, T)
C--------------------------------------------------------
C      程序段 A
C      建立输出缓冲
       call engOutputBuffer (ep, p)
C      计算并输出 MATLAB 的显示
       call engEvalString (ep, 'D = .5. * (-9.8) . * T. ^ 2')
       print * , p
C--------------------------------------------------------
C      绘图操作
       call engEvalString (ep, 'plot (T, D);')
       call engEvalString (ep, 'title ("Position vs. Time")')
       call engEvalString (ep, 'xlabel ("Time (seconds)")')
       call engEvalString (ep, 'ylabel ("Position (meters)")')
C      流程选择
       print * , 'Type 0 <return> to Exit'
       print * , 'Type 1 <return> to continue'
       read ( * , * ) temp
       if (temp. eq. 0) then
          print * , 'EXIT!'
          stop
       end if
C      关闭图形窗口
       call engEvalString (ep, 'close;')
C      获取数据并输出
       D = engGetMatrix (ep, 'D')
       call mxCopyPtrToReal8 (mxGetPr (D), dist, 10)
       print * , 'MATLAB computed the following distances:'
       print * , ' time (s) distance (m)'
       do 10 i=1, 10
          print 20, time (i), dist (i)
20     format (' ', G10. 3, G10. 3)
10     continue
C      释放内存并关闭引擎
       call mxFreeMatrix (T)
       call mxFreeMatrix (result)
```

```
      stat = engClose (ep)
C
      stop
      end
```

读者仔细阅读以上程序将会发现，该程序与 fengdemo. f 最大的差别在于程序段 A。在该程序段中，程序为 MATLAB 输出建立了缓冲，并将函数 engEvalString 所包含命令中的分号去除，以使 MATLAB 产生输出，并将缓冲p 中的内容通过print 输出到屏幕上。程序执行时将在屏幕上显示如下内容：

```
D =
    Columns 1 through 7
    -4.9000   -19.6000   -44.1000   -78.4000   -122.5000   -176.4000
-240.1000
  Columns 8 through 10
  -313.6000 -396.9000 -490.0000

Type 0 <return> to Exit
Type 1 <return> to continue
```

7. engPutFull

功　能：向 MATLAB 引擎写入一个满存储类型的 mxArray 结构体。

语　法：integer * 4 function engPutFull (ep, name, m, n, pr, pi)

integer * 4 ep, m, n, pr, pi

character * (*) name

说　明：函数 engPutFull 拥有六个输入参数，含义分别如下：

- ep 为引擎指针
- name 为用户希望写入的 mxArray 结构体的名字
- m 为用户希望写入的 mxArray 结构体的行数
- n 为用户希望写入的 mxArray 结构体的列数
- pr 为用户希望写入的 mxArray 结构体的实数部分的指针
- pi 为用户希望写入的 mxArray 结构体的虚数部分的指针

如果在引擎的工作空间中不存在同名的 mxArray 结构体对象，函数将创建一个新的 mxArray 结构体对象，如果存在，函数将覆盖原有的 mxArray 结构体对象。此外还必须明确一点，函数 engPutFull 只能对存储类型为满的阵列对象进行操作。

该函数在后续的版本中，将被列为过时的函数，希望用户尽可能地不要使用该函数，可以通过组合使用引擎函数 engPutMatrix 和 mx-函数 mxSetM、mxSetN、mxSetPr、mxSetPi 来完成该函数的功能。

8. engPutMatrix

功　能：向 MATLAB 引擎写入一个满存储类型的 mxArray 结构体。

语　法：integer * 4 function engPutMatrix（ep，mp）

integer * 4 mp，ep

说　明：通过引擎函数 engPutMatrix，用户可以向 MATLAB 引擎的工作空间中写入一个 mxArray 结构体类型的变量，输入参数 ep 用于指定 MATLAB 引擎，为一个指针变量，name 用于确定希望写入的 mxArray 结构体的名字，为字符串变量。如果在引擎的工作空间中不存在同名的 mxArray 结构体对象，函数将创建一个新的 mxArray 结构体对象，如果存在，函数将覆盖原有的 mxArray 结构体对象。若函数执行成功，将返回 0，否则返回 1。同时必须非常注意一点，在退出引擎程序前必须对函数新分配的 mxAray 结构体对象进行释放操作。

举　例：参见 6.1.2 节程序 fengdemo.f。

第7章　客户机/服务器应用程序

随着Internet的广泛深入和应用程序开发的模块化趋势，客户机/服务器这种应用程序开发模式越来越受到人们的重视，以至于相当多的软件都提供了这方面的支持，MATLAB当然也不例外。在本章中，我们将对MATLAB这方面的功能进行全面的讲解，第一节主要讲述ActiveX的概念，第二节则讲述MATLAB对ActiveX的支持，第三节将对动态数据交换进行讲述。

7.1　ActiveX的基本概念

在本节中，我们将对ActiveX术语的由来以及ActiveX组件的类型进行介绍。

7.1.1　ActiveX的诞生

ActiveX一词最早是由Microsoft在1996年3月的Internet专业人员研讨会（Internet PDC）上提出，当时ActiveX指的是大会口号“Activate the Internet”，仅仅是一种号召而远非具体的应用程序开发技术和体系结构。

在Internet PDC期间，Microsoft与Netscape针对Internet Web浏览器市场的控制权展开了面对面的交锋，流露了对浏览器市场的极大兴趣，同时Microsoft还展示了从电子存储前端到新型的OLE控件以及虚拟实境交谈等一系列软件和工具。

瞬时间，ActiveX成了Microsoft的新的企业口号，就好像20世纪90年代初的OLE那样，而且在很短的时间内，其含义就远远超出了“Activate the Internet”，它成了定义Web页面到OLE（Object Linking and Embedding，对象链接和嵌入）控件的所有内容的核心术语。一方面，它意味着用户可以通过一系列的小型的、快速的、可重用的组件将自身与Microsoft、Internet和业界开发的新技术紧密相连；另一方面，ActiveX代表了一种Internet和应用程序集成的开发策略。如今，如果哪一个公司或产品在它们的术语中没有出现Internet和ActiveX，无论是从外部还是从内部，那么就意味着它们落伍了。事实上，描述ActiveX就像试图描述什么是红色一样，它不是一种技术或体系结构，而是一种概念和潮流。

7.1.2　ActiveX、OLE和Internet

ActiveX和OLE从某种意义上说，已经成为同义词，人们以前所说的OLE控件（OCXs）现在已被称为了ActiveX控件，OLE DocObjects称为了ActiveX文档，并且一些有关如何实现OLE技术的文档已被更新为ActiveX技术，而仅仅是更换了OLE一词。

但是事实上，情况并非如此的简单，ActiveX并不是为了替换OLE，而是基于COM——小型快速可重用组件（Component Object Model），将它扩展为更加适应Internet、Intranet、商业应用程序和家用应用程序的开发，并且提供了相应的开发工具。

7.1.3　ActiveX 组件的类型

ActiveX 组件的开发，可以分为以下六种类型：

- 自动化服务器
- 自动化控制器
- ActiveX 控件
- COM 对象
- ActiveX 文档
- ActiveX 容器

下面我们将对这六种类型进行简单的讲述。

1. 自动化服务器（Automation Servers）

自动化服务器是一种可以由其他应用程序编程驱动的组件。自动化服务器至少包含一个或多个可由其他应用程序创建或连接的基于 IDispatch 的接口。一个自动化服务器可以没有用户界面（User Interface）也可以拥有，这取决于服务器的特性和功能。。

自动化服务器的运行方式可分为三种，如下：

- 进程内（in-process），即在控制器的运行空间内运行；
- 本地（local），即在服务器自身的进程空间内运行；
- 远地（remote），即在另一台机器的进程空间内运行。

服务器的特定实现方式决定了它将如何以及在何处运行，但也并非绝对如此，例如一个 DLL 既可以为进程内运行，也可以为本地运行和远地运行，而 exe 只能在本地或远程运行。一般来说，进程内运行的自动化服务器运行速度最快，本地次之，远地最差。

2. 自动化控制器（Automation Controllers）

自动化控制器是那些使用和操纵自动化服务器的应用程序，如 DLL 或 EXE，它们不但可以在进程内访问自动化服务器，而且可以以本地或远程方式访问自动化服务器。典型的应用程序包括 Microsoft Excel，Microsoft Word 以及 Visual Basic，例如通过 Visual Basic 的编程语言，用户可以方便地生成、使用和消除自动化服务器，就好像它们是有机的组成部分一样。

3. ActiveX 控件

ActiveX 控件等价于以前的 OLE 控件（OCXs）。一个典型的控件包括设计时和运行时的用户界面，惟一的 IDispatch 的接口定义控件的方法和属性，惟一的 IConnection-Point 接口由于控件可引发的事件。除此之外，一个控件还可以包含对其整个生命周期的一致性支持，以及对剪贴板、拖放等用户界面特性的支持。从结构上看，一个控件有大量必须支持的 COM 接口，以便利用这些特性。

ActiveX 控件永远都是在其所放置的容器中运行，控件的典型扩展名为 OCX，但是从运行的角度来看，它不过是一个标准的 Windows 动态链接库（DLL）。

4. COM 对象

COM 对象在结构上于自动化服务器和自动化控制器类似，它们拥有一个或多个 COM 接口，由很少或根本没有用户界面。然而，COM 对象不能像自动化服务器那样被典型的自动化控制器应用程序所使用，为了使用它们，控制器必须具有希望连接接口的特定知识，这往往由 COM 对象的说明文档提供。在 Windows 98 和 Windows NT 操作系统中，包含上百个 COM 对象和自定义接口，用于对操作系统进行扩展，包括了从控制桌面的外观到 3D 图像的着色等各个方面。COM 对象是一种组织相关功能和数据的良好方式，同时它还保持了 DLL 的高速性能。

5. ActiveX 文档

ActiveX 文档，即以前所说的 OLE DocObject，表示一种不仅仅是简单控件或自动化服务器的对象，它可以是从电子表格到财务应用程序中全部发票的任何东西。与控件一样，文档也有用户界面并包含于容器应用程序中，Microsoft Excel 和 Microsoft Word 就是 ActiveX 文档服务器的典型例子，而 Microsoft Office Binder 和 Microsoft Internet Explorer 则就是 ActiveX 文档容器的典型例子。

ActiveX 文档结构上是对 OLE 链接和嵌入模型的扩展，并对其所在的容器具有更多的控制权。一个显著的变化是对菜单的显示方式，一个典型的 OLE 文档的菜单将会与容器的菜单合并成为一个新的菜单集，而 ActiveX 文档将替换整个容器的菜单系统，只表现出文档的特性，而不是文档与容器的共同特性。文档特性的表达方式是 ActiveX 文档和 OLE 文档所有差别的前提。容器只是一种宿主机制，而由文档本身进行所有控制。另外一个显著的变化是打印和存储。一个 OLE 文档被认为是其容器文档的一部分，因此是作为宿主容器文档的一部分进行打印和存储的，而 ActiveX 文档自身具有打印和存储功能，而不是集中在容器文档中。

ActiveX 文档在一个统一的表示结构中使用，而不是位于嵌入式文档结构中，后者是 OLE 文档的基础，Microsoft Internet Explorer 是 ActiveX 文档方面的一个典型的例子，Explore 只是将 Web 页面展示给用户，但它是作为一个单一的实体进行显示、打印和存储的；而 Microsoft Word 和 Microsoft Excel 则是 OLE 文档结构方面的典型例子，如果将一个 Excel 电子表格嵌入在一个 Word 文档中，电子表格实际上是存储在 Word 文档中，并成为 Word 文档的一个集成部分。

6. ActiveX 容器

ActiveX 容器是一个可以作为自动化服务器、控件、和 ActiveX 文档宿主的应用程序，例如 Visual Basic 和 ActiveX Control Pad 就是 ActiveX 容器的典型的例子，它们可以作为自动化服务器和控件的宿主，此外 Microsoft Office Binder 和 Microsoft Internet Explorer 也是极为典型的 ActiveX 容器的例子，它们不但可以作为自动化服务器和控件的宿主，而且可以作为 ActiveX 文档的宿主。

7.1.4 小结

总的来说，ActiveX是一种基于Microsoft Windows操作系统的组件集成协议，通过ActiveX，开发者和终端用户可以选择由不同的开发商发布的面向应用程序的ActiveX组件，并将它们无缝地集成到自己的应用程序中，从而完成特定的目的。例如，开发一个独立的应用程序，要求具有数据库访问功能、数值分析功能以及商用图形功能，如果开发者全部自行编写，显见任务是非常繁重的，但是基于ActiveX组件，开发者就可以通过一个开发商选择数据库访问组件，而通过另一个开发商选择数值分析组件，再通过第三个开发商选择商用图形组件，并最终将它们集成在一起，这样应用程序就开发完毕了，这样不但提高了开发效率，而且肯定更加易于使用。

要理解ActiveX必须非常注意两个概念，即COM和ActiveX接口。COM，即微软组件对象模型Component Object Model，它是所有ActiveX组件的基础，它定义了基本的ActiveX组件的模型；ActiveX接口，即ActiveX Interface，是所有ActiveX组件的基本的组成部分，每一个ActiveX组件至少拥有一个或多个命名的接口，每一个接口是一系列相关的方法、属性和事件的集合。方法非常类似于函数，调用它们是为了要求组件完成一定的功能或动作；属性是声明的由组件支持的变量，描述了组件在一定时刻的状态，例如文字的颜色、文件的名字等；事件是有外界激发组件时，组件相应产生的动作。

此外，COM的一个重要特性就是它支持多接口，其中一些为标准接口，它们被定义为ActiveX的组成部分，而另一些为用户自定义的接口，由各个开发商定义。为了使用ActiveX组件，用户必须清楚地知道各组件所定义的自定义接口及其方法、属性和事件，这些信息通常由开发商提供。

至此，我们已经对ActiveX术语和ActiveX组件的类型进行了简单的介绍，目的是为了方便后续的讲解，但是内容非常不全面，希望进一步了解ActiveX的读者请自行参阅相关的书籍。

7.2 MATLAB ActiveX集成

在MATLAB中，对两种类型的ActiveX技术提供了支持，即ActiveX容器和ActiveX自动化，其中ActiveX自动化包含了ActiveX自动化服务器和ActiveX自动化控制器两种类型的ActiveX组件。通过MATLAB ActiveX自动化服务器技术，用户可以在自己编写的ActiveX自动化控制器程序或ActiveX容器程序中对MATLAB进行操纵，如在MATLAB的工作空间中执行一定的命令，向MATLAB工作空间输入矩阵数据和从MATLAB工作空间获取矩阵数据；而通过MATLAB ActiveX自动化控制器技术，用户可以在MATLAB中，通过编写M函数对ActiveX自动化化服务器进行各种控制，包括初始化和删除。此外通过MATLAB ActiveX容器技术，允许用户在MATLAB中使用大量其他的ActiveX控件。相关概念请读者参见7.1节。

这里必须非常注意对几个概念的区别：第一，ActiveX自动化控制器所完成的功能只是ActiveX容器所完成功能的一个子集；第二，所有的ActiveX控件从某种程度上都可以认为是ActiveX自动化服务器，但是并非所有的ActiveX自动化服务器都可以认为是

ActiveX 控件，它们最主要的区别在于，一般来说，非 ActiveX 控件的 ActiveX 自动化服务器不可以被作为客户端应用程序的一部分有机和可视地嵌入到应用程序，例如 MATLAB 就是一个非常典型的例子，它是一个应用程序自动化服务器，而不是一个应用程序控件，因为它不能被嵌入到客户端的应用程序中。

在本节中，我们将分别从客户端和服务器端对 MATLAB 的 ActiveX 功能进行必要的介绍。

7.2.1 MATLAB ActiveX 自动化控制器

MATLAB 作为 ActiveX 自动化控制器时，就相当于一个客户端的应用程序，如果希望在 MATLAB 中使用任何的 ActiveX 组件，首先必须清楚地知道该 ActiveX 组件的名字，即 ProgID，其次必须清楚地知道组件所使用的各个接口的名字、方法、属性和事件，这些信息一般可以通过相关的 ActiveX 组件的文档获得。一旦拥有了这些信息，用户就可以通过 MATLAB 提供的 ActiveX 自动化控制器支持将 ActiveX 组件集成到 MATLAB 的环境中了。例如 MATLAB 自身拥有一个名为 MWSAMP.MwsampCtrl.1 的样例控件，其方法、属性和事件分别如下：

方法：

Redraw——使控件重绘

Beep——使控件鸣叫

AboutBox——显示“about”对话框

属性：

Radius (integer) ——控件中绘制圆的半径

Label (string) ——控件的标题

事件：

Click——当用户点击控件时触发的事件

在 MATLAB 中，用户可以通过实例化一个 MATLAB 的 activex 对象来创建一个 ActiveX 控件或 ActiveX 自动化服务器对象，这主要是通过运行命令 actxcontrol 和命令 actxsever 该 activex 来完成，这两个命令的返回值即为一个 activex 对象，该对象代表了 ActiveX 组件的默认接口。除了以上方法获得 activex 对象外，当一个 ActiveX 组件具有多个接口时，用户也可以使用命令 get 或 invoke 通过已经存在的接口获取新的 activex 对象。MATLAB 系统中，为用户提供了功能全面的构造 activex 对象和针对 activex 对象操作的命令，详见表 7.1。

表 7.1　activex 对象的构造和操作命令

命　令　名	功　　能
actxcontrol	建立一个 ActiveX 控件对象
actxserver	建立一个 ActiveX 自动化服务器对象
set	对接口的某一属性进行设置
get	从接口获取某一属性的值
invoke	激活接口的一个方法

续表 7.1

命　令　名	功　　能
propedit	要求控件显示其内建的属性页
release	释放一个 activex 对象
send	显示事件列表
delete	删除一个 activex 对象

下面我们对这些命令进行全面的介绍。

1. actxcontrol

功　能：在一个图形窗口内构造一个 ActiveX 控件。

语　法：h = actxcontrol (progid [, position [, handle...
　　[, callback | {event1 eventhandler1;...
　　event2 eventhandler2;...}]]])

说　明：通过命令 actxcontrol，用户可以在一个指定的图形窗口的特定的位置构造一个 ActiveX 控件，并且得到一个返回的 MATLAB activex 对象，该对象代表了所构造的 ActiveX 控件的默认接口。如果用户指定的图形窗口为不可见，那么控件也不可见。当命令返回的 MATLAB activex 对象不需要被继续使用时，必须使用 release 命令对该对象占用的资源和内存进行释放，否则将导致资源浪费。但是必须注意一点，释放接口并不意味着删除。命令的各参数含义如下：

- progid 为一个包含用户希望创建控件名字的字符串，一般该字符串由控件的供应商提供；
- position 为一个矢量，其构成形式为 [x, y, xsize, ysize]，其中 x 和 y 为控件的起始坐标，xsize 和 ysize 为控件的宽和高，单位均为像素，默认取值为 [20, 20, 60, 60]；
- handle 为用户指定的图形窗口的句柄，控件就是在该图形窗口中构造；
- callback 为一个 M 函数的名字，当控件触发了一个事件后，该函数将被调用。该函数可以接收可变数目的输入变量，并且所有的输入参数均被转换为 MATLAB 字符串，其中第一个参数为一个代表被触发事件的数值的字符串，事件的数值由控件定义；
- event 为被触发事件的数值或名字；
- eventhandle 为一个可以接收可变数目输入变量的 M 函数的函数名，当与该函数相联系的控件事件被触发时，该函数将被调用；函数的第一个参数为 activex 对象，而第二个参数为触发事件的数值，这里与 callback 中的 M 函数不同，它不需要将数值转换为字符串形式，同样的是事件的数值由控件定义。

命令中提供了两种对触发事件处理函数的定义方法，用户既可以通过创建

一个单一的 callback 处理函数来响应触发事件，也可以通过构造一个包含被触发事件的数值或名字以及相应的处理函数的单元阵列，对每一个事件给予一个处理函数。单元阵列中，事件及其处理函数的对数不受任何限制。事件的数值或名字用控件自身定义。如果命令执行失败将返回一个 MATLAB 错误信息。

举　例： 1) Callback 形式的事件处理函数的 ActiveX 控件建立：

```
% 指定图形窗口句柄
f = figure ('pos', [100 200 200 200]);
% 在图形窗口中指定位置建立控件
h = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], gcf, 'sampev')
```

其中 MWSAMP. MwsampCtrl. 1 为控件名字，[0 0 200 200] 为控件显示位置，gcf 为当前默认的图形窗口，sampev 为 callback 形式的事件处理函数，其定义如下：

```
function sampev (varargin)

  if (str2num (varargin {1}) == -600);
      disp ('Click Event Fired');
  end
```

2) 单元阵列形式的事件处理函数的 ActiveX 控件建立

```
% 指定图形窗口句柄
f = figure ('pos', [100 200 200 200]);
% 在图形窗口中指定位置建立控件
h = actxcontrol ('SELECTOR.SelectorCtrl.1', ...
                 [0 0 200 200], f, {-600 'myclick'; -601 'my2click'; ...
                 -605 'mymoused'})
```

或者

```
h = actxcontrol ('SELECTOR.SelectorCtrl.1', ...
                 [0 0 200 200], f, {'Click' 'myclick'; ...
               'DblClick' 'my2click'; 'MouseDown' 'mymoused'})
```

其中 SELECTOR. SelectorCtrl. 1 为控件名字，[0 0 200 200] 为控件显示位置，f 为当前的图形窗口句柄，{-600 'myclick'; -601 'my2click'; -605 'mymoused'} 为各事件值及其相对应事件处理函数构成的单元阵列，而 {'Click' 'myclick'; 'DblClick' 'my2click'; 'MouseDown' 'mymoused'} 为各事件名字及其相对应事件处理函数构成的单元阵列。事件处理函数 myclick. m，my2click. m 和 mymoused. m 的定义分别如下：

```
myclick.m:
    function myclick (varargin)
      disp ('Single click function')

my2click.m:
```

```
    function my2click (varargin)
      disp ('Double click function')

mymoused.m:
    function mymoused (varargin)
      disp ('You have reached the mouse down function')
      disp ('The X position is: ')
      varargin (5)
      disp ('The Y position is: ')
      varargin (6)
```

此外用户也可以将所有的事件处理函数放在同一个函数中完成，例如按如下方式建立控件：

```
h = actxcontrol ('SELECTOR.SelectorCtrl.1', ...
          [0 0 200 200], f, {'Click' 'allevents'; ...
         'DblClick' 'allevents'; 'MouseDown' 'allevents'})
```

其中对于所有事件的处理函数均为 allevents.m，其定义如下：

```
function allevents (varargin)
  if (varargin {2} = --600)
  disp ('Single Click Event Fired')
  elseif (varargin {2} = -601)
  disp ('Double Click Event Fired')
  elseif (varargin {2} = -605)
  disp ('Mousedown Event Fired')
end
```

2. actxserver

功　能：开启一个 ActiveX 自动化服务器。

语　法：h = actxserver (progid [, MachineName])

说　明：通过该命令，用户可以将 MATLAB 作为一个客户端应用程序用来开启一个 ActiveX 自动化服务器，并返回一个 activex 对象，该对象代表了所开启的自动化服务器的默认接口。该命令拥有两个输入参数，含义分别如下：

- progid 为一个包含用户希望开启的自动化服务器名字的字符串，一般该字符串由自动化服务器的供应商提供，例如微软为 Microsoft Excel 提供的 progid 为 Excel. Application；
- MachineName 为一个远程运行自动化服务器的主机名。

其中参数 MachineName 为一个可选的参数，并且只能够在支持分布式组件对象模型的环境中使用，既可以为一个 IP 地址，也可以为一个域名。如果命令执行失败将返回一个 MATLAB 错误信息。

本地或远程的自动化服务器不同于 ActiveX 控件，它们运行在另外一个独立的地址空间中，甚至可能在不同的计算机上，不属于 MATLAB 进程的一个

部分，并且自动化服务器所显示的任何用户界面均出现在另外的窗口中，而不与 MATLAB 发生联系，Microsoft Word 就是一种典型的自动化服务器。这里必须注意一点，自动化服务器没有 callback 和事件句柄。

举　例：
```
h = actxserver ('Excel. Application')
set (h, 'Visible', 1);
```

3. set

功　能：用于将接口的某个属性值设定为一个指定的值。

语　法：set (a [, 'propertyname' [, value [, arg1, arg2, ...]]])

说　明：该命令的输入参数含义如下：

- a 为一个 MATLAB activex 对象句柄，该值是由 actxcontrol、invoke、actxserver 和 get 命令执行返回的结果；
- propertyname 为用户希望设定的某个接口属性名；
- value 为希望设定的属性值；
- arg1, arg2, ... argn 为设置属性时可能需要的参数，在某些情况下，属性与函数类似，需要参数。

该命令没有任何返回值。

举　例：
```
% 建立图形窗口
f = figure ('pos', [100 200 200 200]);
% 在图形窗口的指定位置上建立控件
a = actxcontrol ('MWSAMP. MwsampCtrl. 1', [0 0 200 200], f)
% 设置属性值
set (a, 'Label', 'Click to fire event');
set (a, 'Radius', 40);
% 重绘控件
invoke (a, 'Redraw');
```

执行以上这些命令，MATLAB 将显示以下图形（见图 7.1）。

4. get

功　能：从接口获得一个指定属性的值或者获得一个属性的列表。

语　法：v = get (a [, 'propertyname' [, arg1, arg2, ...]])

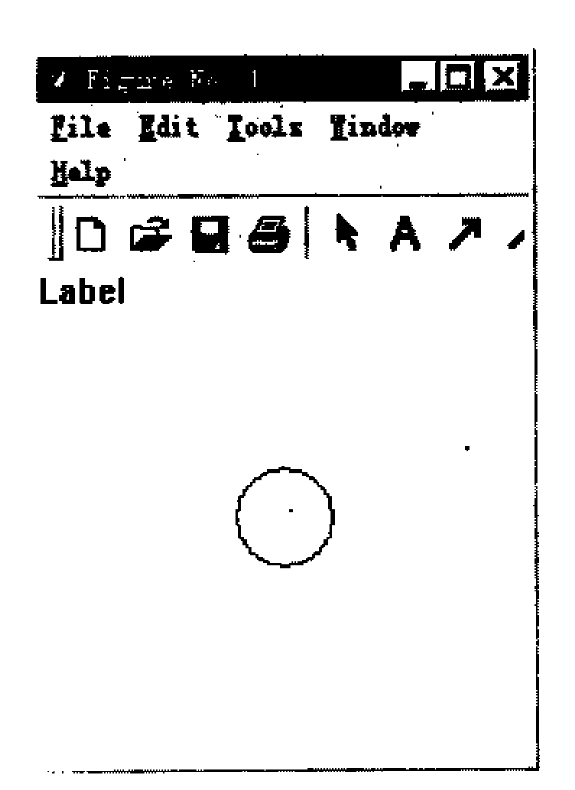
说　明：该命令用于获取接口某个属性的取值，其输入参数的含义如下：

- a 为一个 MATLAB activex 对象句柄，该值是由 actxcontrol、invoke、actxserver 和 get 命令执行后返回的结果；
- propertyname 为用户希望获取内容的接口的某个属性名；
- arg1, arg2, ... argn 为获取属性值时可能需要的参数，在某些情况下，属性与函数类似，需要参数。

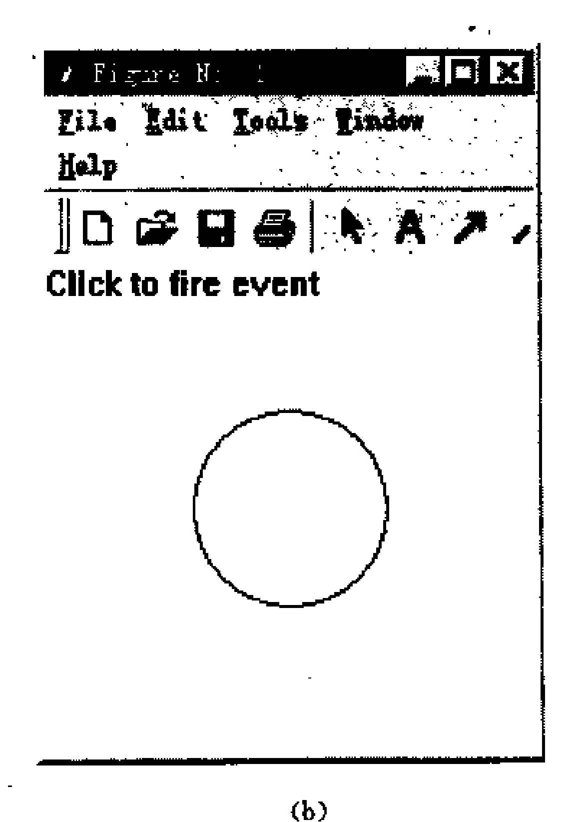
在只给定参数 a 的情况下，命令将列出接口所有属性及其值。

举　例：
```
% 建立图形窗口
```

```
f = figure ('pos', [100 200 200 200]);
% 在图形窗口的指定位置上建立控件
a = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f);
% 获取单个属性值
s = get (a, 'Label')
    s =
    Label
% 获取全部的属性值
get (a)
    Label = Label
    Radius = [20]
```



(a)



(b)

图 7.1　设置属性前后的图形

5. invoke

功　能：激活接口的某个方法并获取返回值或者获取接口的所有方法名。

语　法：v = invoke (a [, 'methodname' [, arg1, arg2, ...]])

说　明：命令 invoke 用于激活接口的某个方法，如果该方法存在返回值的话，则获取该返回值，命令的各参数含义如下：

- a 为一个 MATLAB activex 对象句柄，该值是由 actxcontrol、invoke、actxserver 和 get 命令执行后返回的结果；
- methodname 为用户希望激活的接口的某个方法名；
- arg1, arg2, ... argn 为方法运行可能需要的参数。

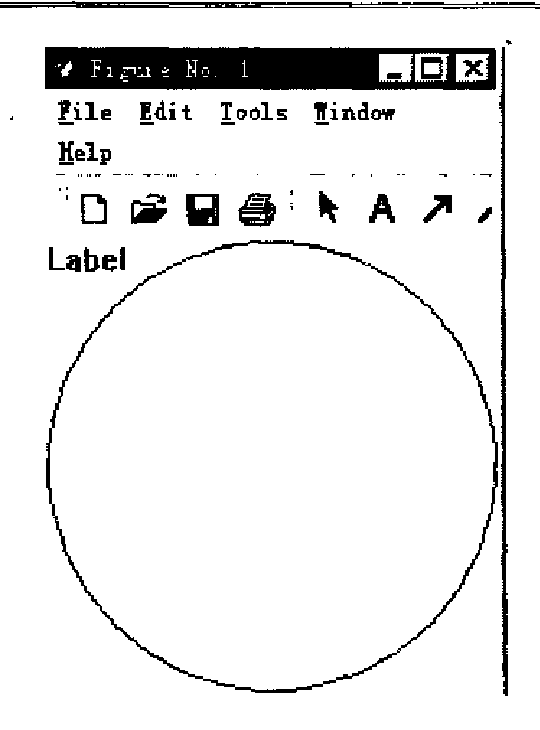


图 7.2　设置属性后重绘的图像

在只给定参数 a 的情况下，命令将列出接口所有的方法。

举　例：% 建立图形窗口

```
f = figure ('pos', [100 200 200 200]);
% 在图形窗口的指定位置上建立控件
a = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f);
% 设置控件属性
set (a, 'Radius', 100);
% 激活 redraw 方法进行控件重绘
v = invoke (a, 'Redraw')
```

执行以上的这些命令后，MATLAB 将显示如图 7.2 所示的图形。

如果仅仅给出参数 a，命令将显示接口所有的方法的定义，如下：

```
invoke (a)
    AboutBox = Void AboutBox ()
    Beep = Void Beep ()
    FireClickEvent = Void FireClickEvent ()
    ……
    SetR8 = Double SetR8 (Double)
    SetR8Array = Variant SetR8Array (Variant)
    SetR8Vector = Variant SetR8Vector (Variant)
```

6. propedit

功　能：要求控件显示其内建的属性页。

语　法：propedit (a)

说　明：该命令要求 ActiveX 控件显示其内建的属性页，如果控件不存在属性页，命

令将执行失败，其输入参数 a 为一个 MATLAB activex 对象，为 actxcontrol、invoke、actxserver 和 get 命令执行后返回的结果。

举　例： % 建立图形窗口

```
f = figure ('pos', [100 200 200 200]);
% 在图形窗口的指定位置上建立控件
a = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f);
% 显示属性页
propedit (a)
```

执行以上的命令后 MATLAB 将显示如图 7.3 所示的图形。

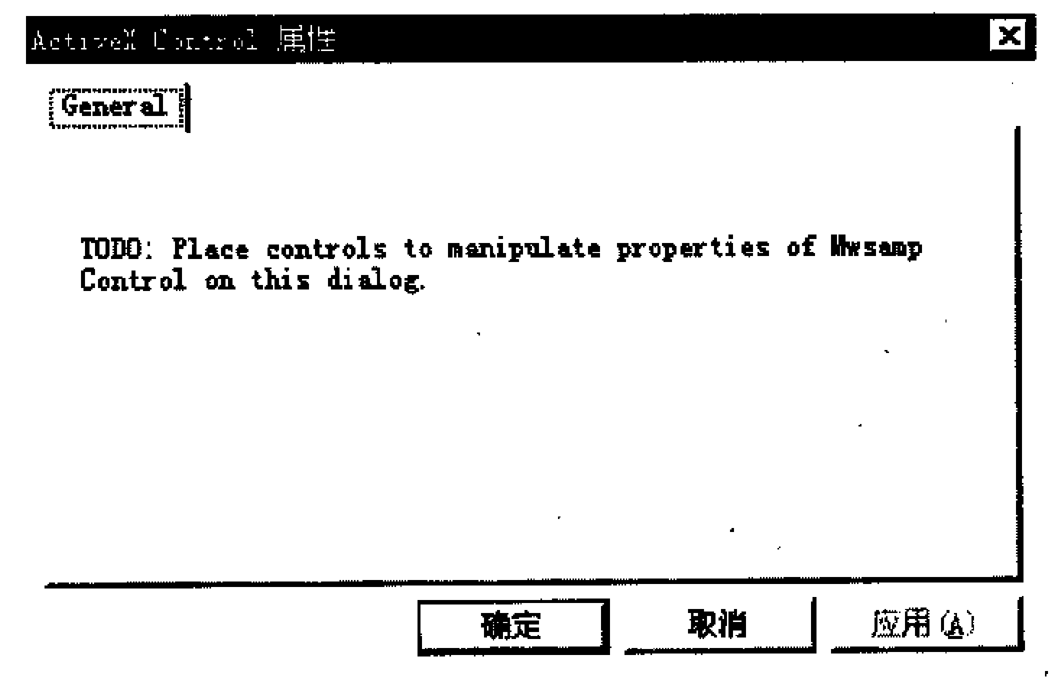


图 7.3　控件的内建属性页

7. release

功　能： 释放一个接口。

语　法： release (a)

说　明： 该命令用来释放接口所占用的所有资源。如果对一个接口使用完毕，必须对其进行释放，否则将导致资源浪费。当对某个接口进行资源释放后，接口将变为无效，所有的对该接口的操作将会报错。输入参数 a 为一个 MATLAB activex 对象，为 actxcontrol、invoke、actxserver 和 get 命令执行后返回的结果。

举　例： % 建立图形窗口

```
f = figure ('pos', [100 200 200 200]);
% 在图形窗口的指定位置上建立控件
a = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f);
% 释放接口
release (a)
```

8. send

功　能：显示一个接口事件列表。

语　法：send (a)

说　明：该命令用于显示控件的所有事件列表，输入参数 a 为一个 MATLAB activex 对象，为 actxcontrol、invoke、actxserver 和 get 命令执行后返回的结果。

举　例：

```
% 建立图形窗口
f = figure ('pos', [100 200 200 200]);
% 在图形窗口的指定位置上建立控件
a = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f);
% 显示事件列表
send (a)
Click = Void Click ()
```

9. delete

功　能：删除一个 ActiveX 控件或自动化服务器。

语　法：delete (a)

说　明：该命令用于删除一个 ActiveX 控件或自动化服务器，其输入参数 a 为一个 MATLAB activex 对象，为 actxcontrol、invoke、actxserver 和 get 命令执行后返回的结果。该命令不同于命令 release，它将释放所有的控件接口或服务器接口，并且删除控件或服务器，而 release 仅仅释放参数中指定的接口。

举　例：

```
% 建立图形窗口
f = figure ('pos', [100 200 200 200]);
% 在图形窗口的指定位置上建立控件
a = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f);
% 释放接口
delete (a)
```

在上面的内容中，我们对构造 activex 对象和操作 activex 对象的命令进行了介绍，比较全面地介绍了 MATLAB 作为客户端对 ActiveX 的支持，这里仍有五点内容需要特别指出：

第一是关于 ActiveX 控件事件处理函数编写。当一个控件希望告知包含它的容器发生了一些其感兴趣的事情时，将触发一个事件，例如许多控件在用户使用鼠标单击它们时，将触发一个 single-click 事件。在 MATLAB 中，当用户使用命令 actxcontrol 创建控件时，可以通过输入参数 callback 或事件-事件处理函数单元阵列，可选地将控件事件与控件事件处理函数联系起来。下面分别对两种方式进行说明。

当使用 callback 方式时，用户只需定义一个事件处理函数，控件在触发任何事件后，都将调用该函数，其一般定义形式为

```
function event_process (varargin)
```

```
        if (str2num (varargin {1} == -600)
          disp ('Click Event Fired');
        ......
    end
```

所有传向该函数的参数均为字符串类型，并且其中第一个输入参数包含了表示被触发事件的数值，该数值也被存储为字符串，因此在使用前必须通过 MATLAB 内建函数 str2num 进行转换，如函数中的第二行。为了能够处理多个事件，用户可以通过多个 if 语句进行判断，并且进行不同的处理，或者使用 switch-case 语句。

当使用事件-事件处理函数单元阵列方式将事件与事件处理函数相联系时，与 callback 完全不同，它需要对每一个事件单独提供一个处理函数，当触发不同的事件时，调用不同的处理函数。例如为鼠标单击事件提供处理函数 click1.m，为鼠标双击事件提供处理函数 click2.m。这种事件处理方式较前一种方式易于理解，并且利于函数的修改。在特殊情况下，用户也可以将所有的事件处理函数设置为同一个函数，这样在发生任何事件时，都将调用该函数，这与使用 callback 方式的效果相同。

第二是关于 MATLAB 中 ActiveX 接口的建立与释放。由于一个 ActiveX 对象可能拥有多个接口，而使用命令 actxcontrol 和 actxserver 获得的接口均为 ActiveX 对象的默认接口，如何获取 ActiveX 对象的其他接口呢？MATLAB 提供了其他的两种方式，即通过命令 get 和命令 invoke 获取，当执行这两个命令的返回值为接口对象时，MATLAB 自动将返回值转化为 activex 对象类型，在 MATLAB 中所有的接口均用 activex 对象表示。一旦实例化了 activex 对象，在使用完毕后一定要记住释放该对象，否则将导致资源的严重浪费。此外，为了避免内存等资源的泄漏，MATLAB 提供了自动的接口释放机制。当存放 ActiveX 控件的图形窗口被关闭或删除时，MATLAB 会自动释放所有的接口；同时在 MATLAB 被关闭时，MATLAB 同样将释放所有的 ActiveX 自动化服务器的接口。

第三是关于 ActiveX 集合(collection)的使用。ActiveX 集合本身是一个特殊的接口，通过这种接口，可以以组的方式对许多相互联系能够重复使用的 ActiveX 对象提供支持。在该接口中，存在一个只读的属性 Count，代表了集合中 ActiveX 对象的数目；此外通过集合接口的 Item 方法，允许用户从集合中获取单个的 ActiveX 对象。通常在使用 Item 方法时，需要提供一个索引，用于指明希望获取哪一个 ActiveX 对象，一般情况下，方法执行完毕后，将得到一个接口对象。索引的数据类型必须与特定的 ActiveX 集合相匹配。下面是一个简单的使用 ActiveX 集合的例子，该例子重复使用集合中所有接口的 Redraw 方法：

```
hCollection = get (hControl, 'Plots');
for i =1: get (hCollection, 'Count')
    hPlot = invoke (hCollection, 'Item', i);
    invoke (hPlot, 'Redraw');
    release (hPlot);
end
release (hCollection);
```

第四是关于 ActiveX 和 MATLAB 之间的数据类型转换。由于 ActiveX 定义了一系

列不同的数据格式和类型，因此在使用它们前，必须知道MATLAB 是如何将这些数据转换到自己的工作空间中。在下列两种情况下，数据的类型必须被转换：

- 数据为由 ActiveX 的属性获得；
- 数据为调用 ActiveX 方法的返回值。

表 7.2 说明了 ActiveX 数据类型向 MATLAB 数据类型转换的关系。

表 7.2　数据转换

ActiveX 数据类型	MATLAB 数据类型
String File Time Error Decimal Date	MaTLAB String
Currency Hresult Int/Unsigned (2, 4, 8) Bool Real (Sigle/Double Precision)	Scalar Double
Null	NaN
Array of 　Currency 　Hresult 　Int/Unsigned (2, 4, 8) 　Bool 　Real (Single/Double Precision)	Matrix of Double
Variant Array of Variant	Cell Array
Idispath	activex Object
Empty Unknown Void Ptr Carray Userdefined Blob Stream Storage Streamed Object Stored Object Blob Object CF	不转换（报错）

第五是关于 MATLAB 作为分布式 COM 服务器客户端的使用。分布式的组件对象模型（DCOM）是一种对象分布处理的机制，它允许 ActiveX 客户端在网络上远程地使用 ActiveX 对象。MATLAB 作为一个 DCOM 服务器，已经在 Windows NT4.0 上测试通过，当其作为一个远程的服务器使用时，MATLAB 将在远程的计算机上启动。

MATLAB 作为一个 ActiveX 容器，虽然对 ActiveX 对象的使用提供了大量的支持，且使用也比较方便，但是与其他的一些 ActiveX 容器相比，仍然存在不足，主要表现在下

面的几点：

- MATLAB 仅仅支持索引的集合；
- MATLAB 不支持事件参数的应用传递；
- MATLAB 不支持从事件处理函数返回值；
- 控件的位置矢量不能被改变或获取；
- ActiveX 控件不能随图形窗口被打印。

7.2.2　MATLAB 自动化服务器

在 Windows 平台上，MATLAB 不但对自动化控制器功能提供了支持，而且还可以作为自动化服务器。当 MATLAB 作为自动化服务器时，它可以被 Windows 平台上的任何可以作为自动化控制器的应用程序所使用，一些典型的自动化控制器包括 Microsoft Excel，Microsoft Access，Visual Basic 和 Visual C++等。通过使用 MATLAB 自动化服务器功能，用户可以在自己的应用程序中执行 MATLAB 命令，并从 MATLAB 的工作空间中获取 mxArray 结构体数据以及向 MATLAB 输送数据。

将 MATLAB 作为一个自动化服务器使用，用户首先必须查阅所希望使用的自动化控制器的文档，查明如何在控制器中开启一个自动化服务器，其次必须知道 MATLAB ActiveX 对象在系统注册表中定义的名字，即 ProgID，一般可以使用 Matlab. Application 或者使用 Matlab. Application. Single，它们分别代表了不同的含义，当应用程序使用 Matlab. Application 作为 ProgID 启动 MATLAB 自动化服务器时，表示将 MATLAB 自动化服务器作为一个共享的服务器，当其他的应用程序同样以 Matlab. Application 作为 ProgID 开启 MATLAB 自动化服务器时，系统将不再另外初始化一个服务器，而是使用同一个服务器来完成所有的请求；而当应用程序使用 Matlab. Application. Single 作为 ProgID 开启 MATLAB 自动化服务器时，表示将 MATLAB 自动化服务器作为一个单独的服务器使用，而不与别的应用程序共享，这时其他的应用程序无论以何种 ProgID 开启 MATLAB 自动化服务器，系统都将重新初始化一个新的 MATLAB 自动化服务器。

知道了这两方面的信息，用户就可以开启 MATLAB 自动化服务器了。需要特别指出的一点是：具体开启 MATLAB 自动化服务器的方法依赖于用户所选择的自动化控制器，但是用来标识 MATLAB ActiveX 自动化服务器的名字却是不会发生变化的。下面是一段 VIsual Basic 的程序源代码，它演示了如何在 Visual Basic 中开启 MATLAB 的自动化服务器功能：

```
Dim MatLab As Object
Set MatLab = CreateObject ("Matlab. Application")
```

其中定义了对象类型数据 Matlab，并且通过函数 CreateObject 以 Matlab. Application 为输入参数，开启了 MATLAB 自动化服务器。当用户完成这些操作后，就可以开始使用 MATLAB 自动化服务器了。

此外除了上面讲述的开启 MATLAB 自动化服务器的方法之外，还可以在启动 MATLAB 时直接将 MATLAB 初始化为一个自动化服务器，只需在启动 MATLAB 时使用参数/Automation 即可。这样在应用程序要求建立一个 ActiveX 连接时，系统将完成它们之间的连接操作，而不会再重新开启一个 MATLAB 自动化服务器了。但是如果在启动

MATLAB 时并没用使用参数/Automation，这时系统将重新初始化一个 MATLAB 的实例，作为自动化服务器。

还有一点值得说明的是，MATLAB 提供了对 DCOM 的支持，应用程序可以通过网络远程启动一个 MATLAB 自动化服务器，用于完成一定的计算功能，从而充分利用网络资源的优势。

对 MATLAB 自动化服务器的使用，必须通过服务器所提供的三个方法，即 Execute，PutFullMatrix 和 GetFullMatrix。通过它们，用户不但可以在 MATLAB 中执行任何合法的命令，而且可以向 MATLAB 输送数据，同时也可以从 MATLAB 中获取数据。下面，我们分别对这三个方法进行说明。

1. Execute

功　能：用于执行一个合法的 MATLAB 命令。

语　法：BSTR Execute （ [in] BSTR Command）

说　明：方法 Execute 的输入参数为一个字符串类型的变量，它可以包含任何的合法的 MATLAB 命令，MATLAB 将执行该字符串所包含的命令，并将结果以字符串的形式进行输出，同时命令所产生的任何图形窗口都将被直接显示在屏幕上。

举　例：下面是一段 Visual Basic 的代码，用于启动 MATLAB 自动化服务器，并且通过方法 Execute 执行一个 MATLAB 绘图命令。执行该段代码后，将显示如图 7.4 所示的图形。

```
Dim MatLab As Object
Dim Result As String
Set MatLab = CreateObject （"Matlab.Application"）
Result = MatLab.Execute （"surf （peaks）"）
```

2. GetFullMatrix

功　能：用于从 MATLAB 的工作空间中获取数据。

语　法：void GetFullMatrix （

```
                [in] BSTR Name,
                [in] BSTR Workspace,
                [in, out] SAFEARRAY （double） * pr,
                [in, out] SAFEARRAY （double） * pi);
```

说　明：通过该方法，用户可以从指定的 MATLAB 的工作空间中获取指定名字的 mxArray 结构体的数据，该结构体既可以为一维也可以为二维，可以为实数类型也可以为复数类型，但是该结构体的存储类型必须为满，即该方法不可以对稀疏类型的矩阵进行数据提取。方法的各输入参数的含义如下：

- name 为用户希望从 MATLAB 的工作空间中提取的 mxArray 结构体的名字；
- Workspace 为用户指定的 MATLAB 工作空间的名字，通常情况下，该

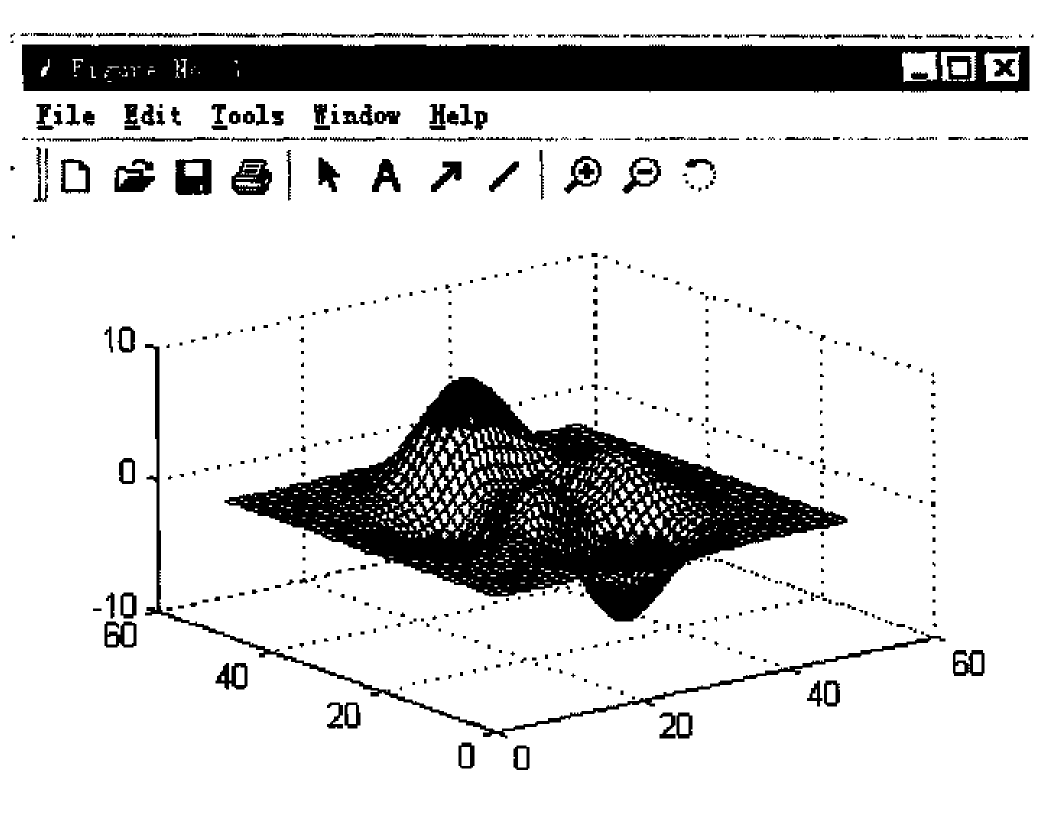


图 7.4　surf（peak）所产生的图形

变量有三种取值：

a）“base”，当变量 Workspace 取该值时，表示从 MATLAB 的默认的工作空间中提取数据；

b）“global”，当变量 Workspace 取该值时，表示从 MATLAB 的全局工作空间中提取数据；

c）“caller”，除了在 MEX 文件中，目前该取值并没有任何意义。

- pr 为一个与用户所希望提取的 mxArray 结构体同样大小的实数阵列的指针。在方法执行完毕后，该指针指向的阵列中将包含用户所希望提取的 mxArray 结构体的实数部分的数据；
- pi 为一个与用户所希望提取的 mxArray 结构体同样大小的实数阵列的指针。在方法执行完毕后，该指针指向的阵列中将包含用户所希望提取的 mxArray 结构体的虚数部分的数据。

举　例：下面是一个使用 Visual Basic 编写的一段范例程序：

```
rem 变量声明
  Dim MatLab As Object
  Dim Result As String
Dim MReal (1, 3) As Double
  Dim MImag () As Double
  Dim RealValue As Double
  Dim i, j As Integer
rem 启动 MATLAB 自动化服务器
```

```
  Set MatLab = CreateObject ( "Matlab.Application")

rem 调用 Execute 方法执行一条赋值命令

  Result = MatLab.Execute ( "a = [1 2 3 4; 5 6 7 8;]")

rem 调用 GetFullMatrix 方法获取数据
  Call MatLab.GetFullMatrix ( "a", "base", MReal, MImag)
rem 执行赋值操作
  For i = 0 To 1
    For j = 0 To 3
      RealValue = MReal (i, j)
    Next j
  Next i
```

3. PutFullMatrix

功　能：用于向 MATLAB 的工作空间中输出一个指定名字的 mxArray 结构体。

语　法：void PutFullMatrix (
　　[in] BSTR Name,
　　[in] BSTR Workspace,
　　[in] SAFEARRAY (double) pr,
　　[in] SAFEARRAY (double) pi);

说　明：通过该方法，用户可以向指定的 MATLAB 的工作空间中输出一个指定名字的 mxArray 结构体，该结构体既可以为一维也可以为二维，可以为实数类型也可以为复数类型，但是该结构体的存储类型必须为满，即该方法不可以向 MATLAB 输出稀疏类型的矩阵。方法的各输入参数的含义如下：

- name 为用户希望向 MATLAB 的工作空间中输出的 mxArray 结构体的名字；
- Workspace 为用户指定的 MATLAB 工作空间的名字，通常情况下，该变量有三种取值：
  a) "base"，当变量 Workspace 取该值时，表示向 MATLAB 的默认的工作空间中输出数据；
  b) "global"，当变量 Workspace 取该值时，表示向 MATLAB 的全局工作空间中输出数据；
  c) "caller"，除了在 MEX 文件中，目前该取值并没有任何意义。
- pr 为一个双精度类型的实数阵列，它包含了用户希望输出到 MATLAB 工作空间中的 mxArray 结构体的实数部分的数据；
- pi 为一个双精度类型的实数阵列，它包含了用户希望输出到 MATLAB 工作空间中的 mxArray 结构体的虚数部分的数据，如果输出到 MATLAB 的 mxArray 结构体为实数类型时，同样必须传送此参数，

不过内容为空。

举　例：下面是一个使用 Visual Basic 编写的一段范例程序：

```
rem 变量声明
  Dim MatLab As Object
  Dim MReal (1, 3) As Double
  Dim MImag () As Double
  Dim i, j As Integer

rem 为数组变量赋值
  For i = 0 To 1
    For j = 0 To 3
      MReal (i, j) = I * j;
    Next j
    Next I

rem 开启 MATLAB 自动化服务器
  Set MatLab = CreateObject ("Matlab.Application")

rem 向 MATLAB 输出数据
  Call MatLab.PutFullMatrix ("a","base", MReal, MImag)
```

7.3　动态数据交换

动态数据交换（DDE），即 Dynamic Data Exchange，是 Window 98 支持的几种进程间通信机制之一，其他三种机制是 Windows 剪贴板、动态链接库中的共享内存和 ActiveX。DDE 的功能没有 ActiveX 那样强大，不过相对于 ActiveX 来说，DDE 较为容易实现。本节中，我们将首先介绍 DDE 的基本概念和术语，然后对 MATLAB 所提供的 DDE 功能进行介绍。

7.3.1　DDE 的基本概念和术语

DDE 是基于 Windows 的消息机制，两个 Windows 应用程序通过相互之间传递消息进行“对话”，这两个程序分别被称为“服务器”和“客户”。DDE 服务器是一个维护着其他 Windows 程序可能使用的数据的程序，而 DDE 客户则是从服务器获得这些数据的程序。

DDE 对话是由客户程序发动的。客户程序将一条称为 WM_DDE_INITIATE 的消息发给当前运行的所有 Windows 程序，这条消息指明了客户程序所需要的数据类别，拥有这些数据的 DDE 服务器可以响应这条消息，这样，一个对话就开始了。

同一个 Windows 应用程序既可以是一个程序的客户，也可以是另一个程序的服务器，但这需要两种不同的对话。一个服务器可以将数据传给多个客户，一个客户也可以从多个服务器获取数据，同样这也需要多种 DDE 对话。一般来说，支持 DDE 的程序将为其所维护的每种对话创建一个隐藏的子窗口。

DDE 对话涉及到的程序不需要为了协同工作而进行特殊的编程，一般来说 DDE 的作者会公开数据是如何被识别的，而 DDE 客户程序的用户就可以利用这些信息来建立两个程序之间的 DDE 对话了。如果要编写两个或多个必须相互通讯而不与其他 Windows 程序进行通讯的 Windows 程序，就可以考虑定义自己的通讯协议。但是如果两个或多个程序既要读共享数据又要写入它，那么就必须将数据存放在一个内存映象文件中。

在客户程序发送的 WM _ DDE _ INITIATE 消息中，不但指明了所需要的数据的类型，而且包含了它所需要的服务器名（service）和主题（topic），其中服务器名标识了客户程序希望建立对话的对象，例如 Microsoft Word 的服务器名为 WinWord，Microsoft Excel 的服务器名为 Excel，而 MATLAB 的服务器名为 Matlab；主题则定义了对话的题目，一般对于建立 DDE 对话的双方均是有意义的，并且主题一般是不区分大小写的。MATLAB 支持的两个主题分别为 System 和 Engine，在后续的内容中，我们将对它们进行说明，对于大多数的应用程序均支持 System 主题，并且支持至少一种其他的主题。主题所支持的数据类型我们也称之为项(item)，一般一个主题支持至少一个项或者更多，项是否敏感大小写，取决于应用程序，例如 MATLAB Engine 主题的项如果是指向矩阵的，那么项就是敏感大小写的，因为 MATLAB 的矩阵名是敏感大小写的。

DDE 通过 Windows 的剪贴板的数据格式对数据进行格式化，然后在应用程序间进行传递。在 MATLAB 作为客户程序时，它仅仅支持文本格式的数据传输，而当 MATLAB 作为服务器应用程序时，却可以支持三种格式的数据传输，分别为文本格式、元文件图（MetaFilepict）格式和 XLTable 格式，分别描述如下：

- 文本格式：以文本格式存放的数据是一个以空字符结束的字符缓冲，缓冲中存在以行存放的数据，则在每一行的结束以一个回车符和一个换行符来标识；如果缓冲中存在以列存放的数据，则在每一列的结束以一个 tab 符来标识。MATLAB 通过支持文本格式来获得远程的 EvalString 命令执行的结果，同时也可以用来请求矩阵数据。此外矩阵数据也可以使用文本格式发送给 MATLAB；
- 元文件图（MetaFilepict）格式：元文件图格式是一种用来存放图形数据的格式，它不同于位图文件通过存放每一个像素点的数据来存放图形的方式，它是记录绘制图形过程中所使用的图形命令和函数的方式来描述图形的。简而言之，元文件图格式是由一系列的二进制形式的编码图形函数的集合。MATLAB 支持元文件图格式是为了获取一些远程执行的图形命令执行的结果；
- XLTable 格式：XLTable 格式是一种剪贴板支持的 Microsoft Excel 使用的数据格式，使用这种数据格式的主要目的是方便与 Microsoft Excel 的数据交换和提高与 Microsoft Excel 数据交换的效率。XLTable 格式是一个拥有数据头的二进制的缓冲，数据头描述了缓冲中存放的数据。

7.3.2　MATLAB 的服务器程序功能

1. 与 MATLAB 服务器通信的原理

一个客户程序可以通过建立 DDE 对话的方法，将 MATLAB 作为一个服务器程序来进行访问。建立 DDE 对话的方法主要有以下几种，具体选用何种方法建立对话，取决于

所使用的应用程序：

- 如果用户使用的应用程序提供了支持 DDE 的函数或者宏，那么用户可以直接使用这些宏和函数来建立与 MATLAB 间的 DDE 对话，典型的应用程序包括 Microsoft Basic，Microsoft Visual C++，Microsoft Word 以及 Microsoft Excel 等；
- 如果用户希望通过自己的应用程序建立与 MATLAB 之间的连接，可以通过使用 MATLAB 引擎函数库或者直接使用 DDE。

图 7.5 描述了应用程序是如何同作为服务器的 MATLAB 通信的原理。两者之间的通信实际上是通过客户应用程序中的 DDE 函数模块与 MATLAB 中的 DDE 服务器模块之间的通信来完成的。客户应用程序中的 DDE 函数模块既可以由应用程序提供，也可以由 MATLAB 的引擎函数库提供。

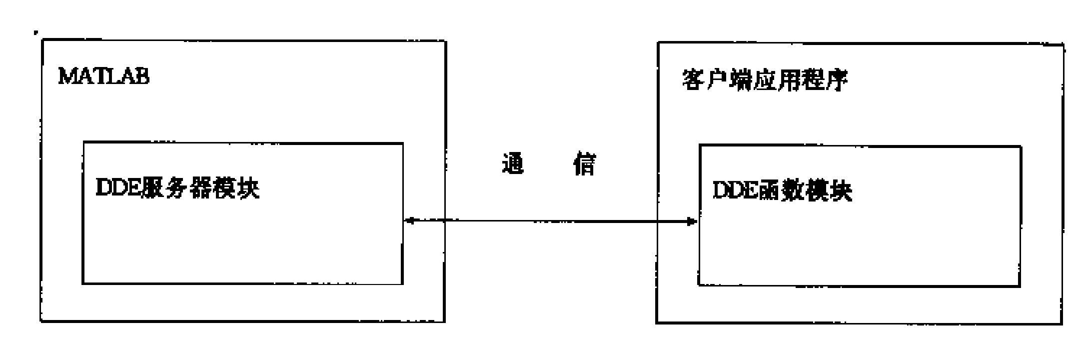


图7.5　MATLAB服务器原理图

2. MATLAB 服务器的构成及说明

在用户将 MATLAB 作为服务器进行访问时，必须提供服务器的名字、主题和项，这是非常必须的。在前面我们已经对 MATLA 提供的主题进行了一定的讲解，下面我们将对 MATLAB 所提的全部 DDE 服务器功能进行说明，图 7.6 为其结构图。

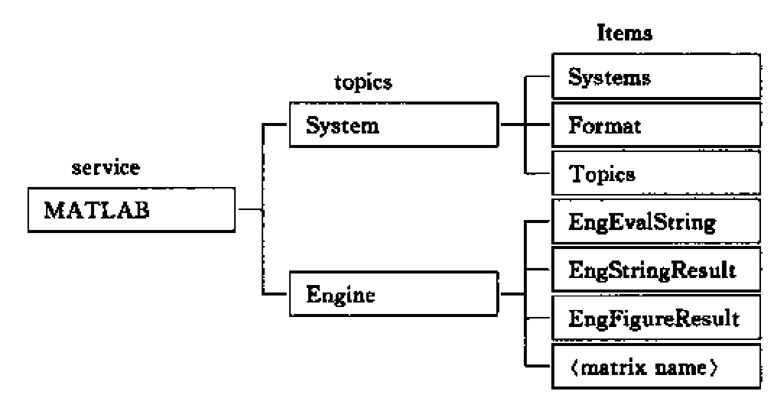


图 7.6　MATLAB DDE 结构图

由上图可以看出，MATLAB 一共支持两种类型的主题，分别为 System 和 Engine：

- System 主题包含了三个项，分别为 SysItems，Format 和 Topics，其中 SysItems 项提供了一个以 tab 字符结尾的包含了 System 主题所支持的全部项的列表；Format 项提供了一个以 tab 字符结尾的包含 MATLAB DDE 服务器支持的全部数据格式名的字符串的列表，MATLAB 服务器共支持三种类型的数据格式，分别为文本格式、元文件图格式和 XLTable 格式，关于这些格式在前面我们已经进

行了讲述；而 Topics 项提供了以 tab 字符结尾的包含 MATLAB DDE 服务器支持的全部主题名的列表。通过它们，用户可以浏览服务器所提供的主题的列表、System 主题提供的项的列表和服务器支持的格式的列表。

- Engine 主题对三种客户端应用程序中可能的 DDE 操作提供了支持，这些操作包括发送命令到 MATLAB 中执行、从 MATLAB 中请求数据和向 MATLAB 发送数据：
  - 发送命令到 MATLAB 中执行：客户端应用程序可以通过 DDE 执行操作向 MATLAB 发送命令进行执行。在 MATLAB 的引擎主题中，支持两种形式的 DDE 执行操作，这是因为一些客户端应用程序在执行 DDE 执行操作时不但要求用户提供项的名字，而且还要求用户提供相应的希望执行的 MATLAB 命令的名字，而一些客户端应用程序只要求用户提供希望执行命令的名字。在两种形式中，命令名必须以文本格式进行存放。对于绝大多数的客户程序来说，它们均以文本格式为默认的传送 DDE 执行命令的格式，也就是说，如果用户不清楚具体的数据格式，那么在一般情况下应该为文本格式。
  - 从 MATLAB 中请求数据：客户端应用程序可以通过 DDE 请求操作向 MATLAB 服务器请求数据。在 MATLAB 的引擎主题中，支持三种类型的数据请求操作，分别为文本格式数据的请求操作、图形数据的请求操作和指定矩阵的数据请求操作：
    - ▲ 通过使用文本格式数据的 EngStringResult 项，用户可以请求 DDE 执行命令的字符串结果；
    - ▲ 通过 EngFigureResult 项，用户可以请求一个 DDE 执行命令的图形结果；这里必须注意的是 EngFigureResult 项可以配合两种格式的数据进行使用，即文本格式和元文件图格式。当使用文本格式的数据时，命令执行返回的结果为字符串“yes”或“no”，如果返回的字符串为“yes”，则表示当前图形的元文件已经被存放于剪贴板中；如果返回的字符串为“no”，则表示剪贴板中没有数据。这项功能主要是提供给那些智能接收文本格式数据的客户端程序使用，例如 Microsoft Word。当使用元文件图格式的数据时，EngFigureResult 项将直接返回包含当前图像的元文件。
    - ▲ 当用户希望请求某个矩阵的数据时，可以直接将该矩阵的名字作为项传递给 MATLAB 服务器，这时可以使用两种不同的数据格式，分别为文本格式和 XLTable 格式。
  - 向 MATLAB 发送数据：客户应用程序通过使用 DDE 发送操作可以向 MATLAB 发送数据。在 MATLAB 的引擎主题中，支持两种类型的 DDE 传送操作，即在 MATLAB 的工作空间中创建新的矩阵和更新已经存在的矩阵，项用来表示用户希望创建或更新的矩阵名。MATLAB 服务器通过判断用户确定的矩阵名在工作空间中是否存在，来决定是创建操作还是更新操作。用户传递给 MATLAB 的数据既可以存放为文本格式，也可以存放为 XLTable 格式。

3. MATLAB DDE 服务器的使用举例

为了演示 MATLAB DDE 服务器的使用，MATLAB 提供了一个典型的 Visual Basic 同 MATLAB DDE 服务器通信的程序。下面我们一步一步地引导用户建立该程序。

首先，启动 Visual Basic，进入集成工作环境，创建一个新的工程；

其次，在 Form1 窗体上，放置两个编辑框控件，并且分别将名字改为 TextInput 和 TextOutput，参见图 7.7；

再次，选择 TextInput 编辑框控件，单击鼠标右键，将弹出一个浮动的菜单，选择其中的 View Code 菜单项，这时将出现代码编辑框；

第四步，从代码编辑框上方的下拉式选项框中选择 KeyPress 事件，这时 Visual Basic 将自动产生事件处理函数的框架，如下：

```
Private Sub TextInput _KeyPress (KeyAscii As Integer)
```

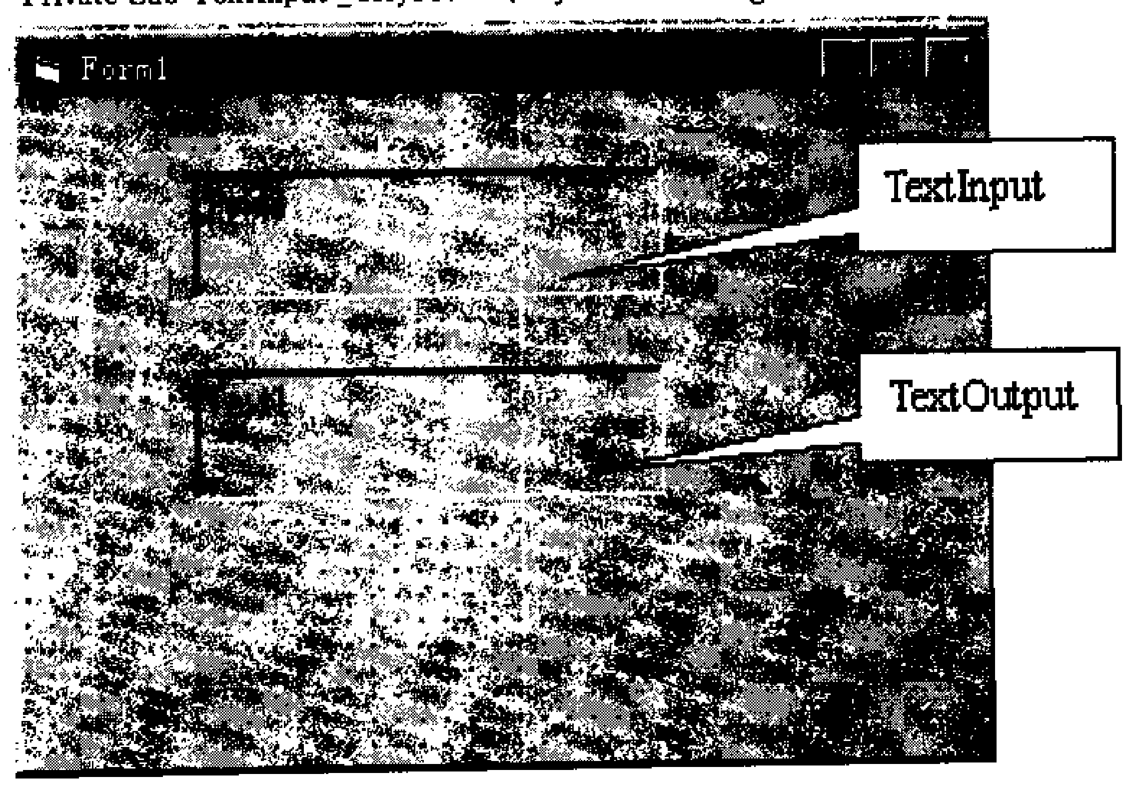


图 7.7　窗体样式

```
    End Sub
```

用户可以在它们之中添加代码，完成事件处理任务；

事件处理的代码如下：

```
Rem  判断输入的字符是否为回车，如果是，则进行后续的操作，
     如果不是则直接返回

If KeyAscii = vbKeyReturn Then
    Rem 如果用户键入回车符，执行前面的操作

    Rem 首先断开控件 TextInput 与其他所有 DDE 服务器的连接
```

```
TextInput.LinkMode = vbLinkNone

Rem 将控件 TextInput 的 DDE 连接对象和主题设置为 MATLAB|Engine
TextInput.LinkTopic = "MATLAB|Engine"

Rem 将控件 TextInput 的 DDE 连接使用的项设置为 EngEvalString
TextInput.LinkItem = "EngEvalString"

Rem 建立连接
TextInput.LinkMode = vbLinkManual

Rem 执行控件 TextInput 中包含的命令
szCommand = TextInput.Text
TextInput.LinkExecute (szCommand)

Rem 断开控件 TextInput 与 MATLAB 服务器的连接
TextInput.LinkMode = vbLinkNone

Rem 首先断开控件 TextOutput 与其他所有 DDE 服务器的连接
TextOutput.LinkMode = vbLinkNone

Rem 将控件 TextOutput 的 DDE 连接对象和主题设置为 MATLAB|Engine
TextOutput.LinkTopic = "MATLAB|Engine"

Rem 将控件 TextOutput 的 DDE 连接使用的项设置为 EngStringResult
TextOutput.LinkItem = "EngStringResult"

Rem 建立连接
TextOutput.LinkMode = vbLinkManual

Rem 获取数据
TextOutput.LinkRequest

Rem 断开控件 TextOutput 与 MATLAB 服务器的连接
TextOutput.LinkMode = vbLinkNone
End If
```

其完成的主要功能为：当用户在编辑框 TextInput 中键入一个 MATLAB 命令并回车后，将该命令传送给 MATLAB 执行，然后将 MATLAB 的输出结果显示在编辑框 TextOutput 中。图 7.8 为该程序执行的结果。

这里必须非常注意一点，也是与 MATLAB 引擎极为不同的一点是在执行该程序时，MATLAB 必须已经处于开启状态，这主要是因为 DDE 只是一种进程间通信的手段，它不负责对为开启的进程进行启动。如果在执行该程序之前，没有启动 MATLAB，程序将报错，如图 7.9 所示。

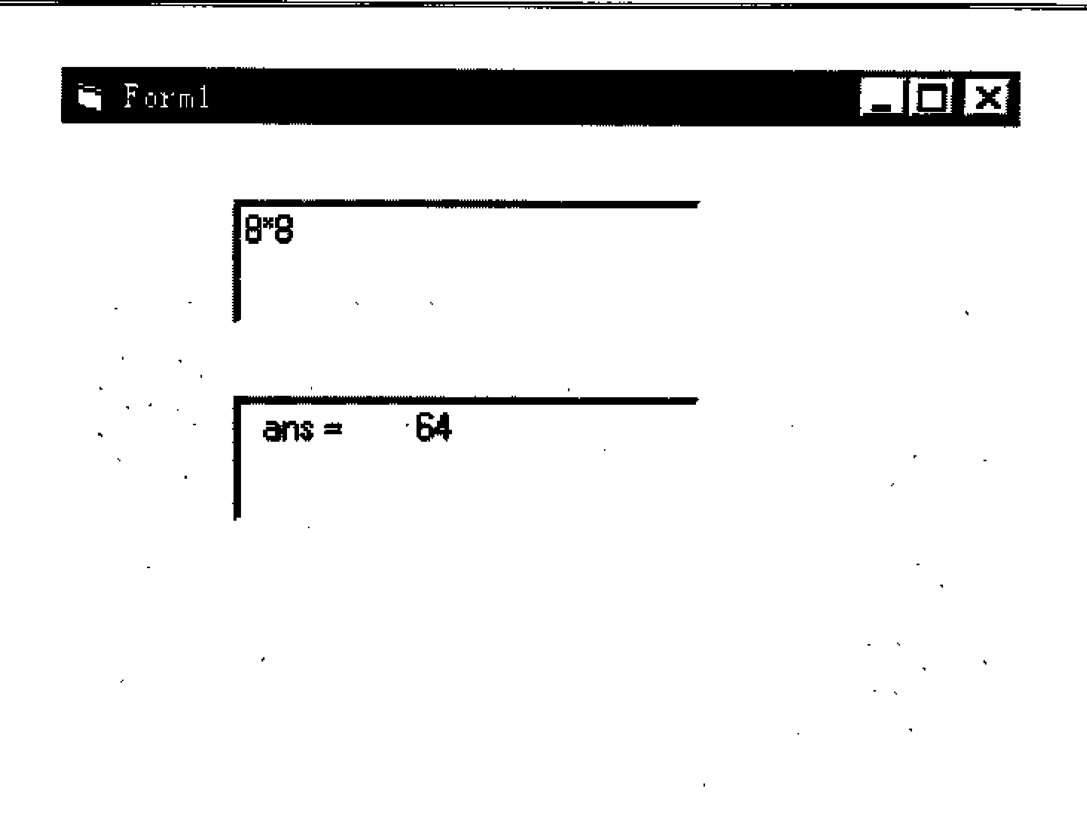


图 7.8　执行结果

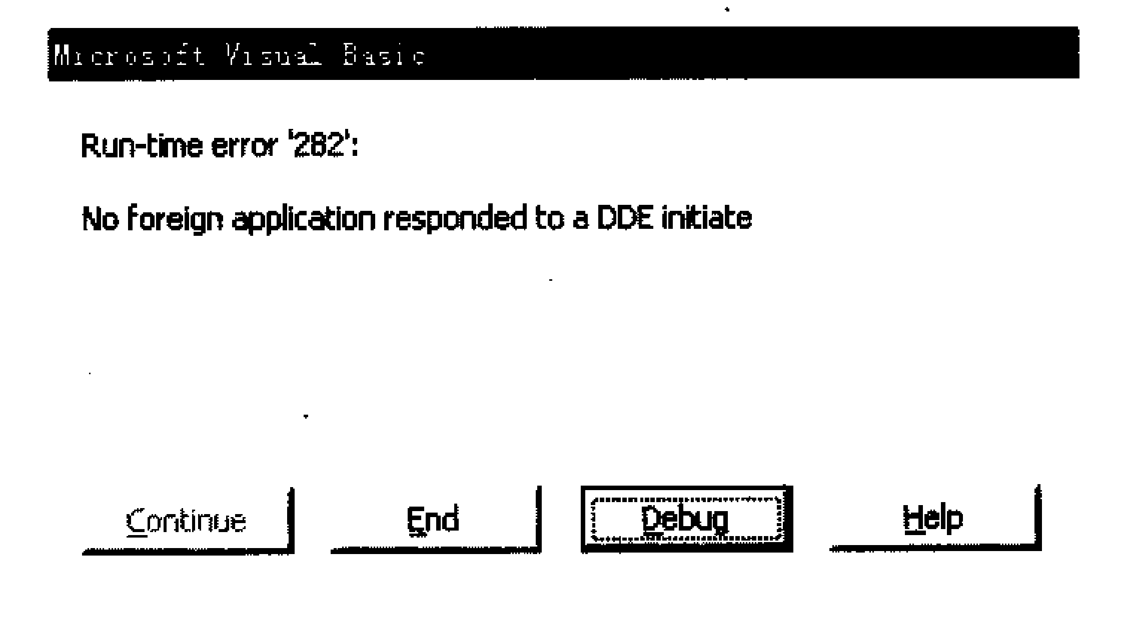


图 7.9　错误消息

7.3.3　MATLAB 的客户端程序功能

MATLAB 通过自身的 DDE 客户端模块与服务器应用程序的 DDE 服务模块进行通信，其原理图如图 7.10 所示

MATLAB 中的 DDE 客户端模块包含了一系列的函数（见表 7.3），通过这些函数，用户可以方便地将 MATLAB 作为 DDE 的客户端与服务器应用程序相连接。

表 7.3　DDE 客户端函数

函 数 名	功　　能
ddeadv	在 MATLAB 同 DDE 服务器应用程序之间建立一个顾问连接
ddeexec	向 DDE 服务器应用程序发送用于执行的命令
ddeinit	初始化 MATLAB 同另一个应用程序间的 DDE 对话
ddepoke	从 MATLAB 向 DDE 服务器应用程序发送数据
ddereq	从 DDE 服务器应用程序请求数据
ddeterm	终止 DDE 服务器应用程序与 MATLAB 间的 DDE 对话
ddeunadv	释放 MATLAB 同 DDE 服务器应用程序之间建立的顾问连接

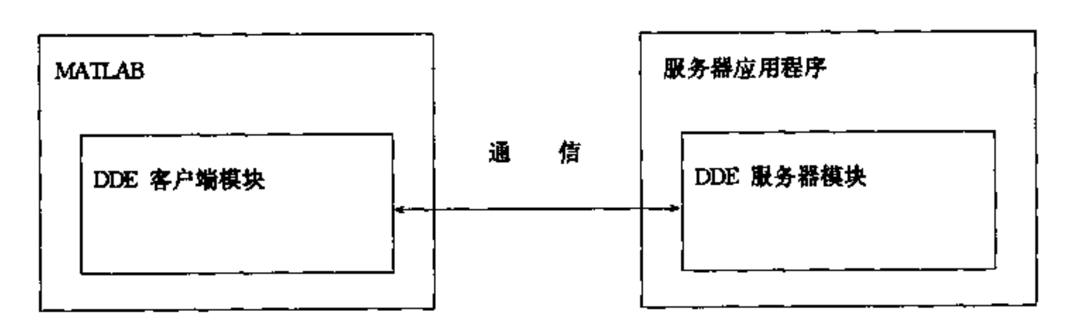


图7.10　MATLAB服务器原理图

理解了这些函数，也就理解了 MATLAB 的客户端功能，下面我们将对这些函数进行逐一的讲解。

1. ddeinit

功　能：初始化 MATLAB 同另一个应用程序间的 DDE 对话。

语　法：channel = ddeinit (service, topic)

说　明：函数 ddeinit 的执行需要两个输入参数，一个为服务器应用程序的标识名 service，另一个为对话主题的标识名 topoic。如果函数执行成功，将返回一个标识通道的值 channel，该值可以为其他的 MATLAB DDE 函数所使用。

举　例：% 初始化一个 MATLAB 同 Microsoft Excel 间的 DDE 对话
% 对象为 Microsoft Excel 的文件 forecast.xls
channel = ddeinit ('excel', 'forecast.xls');

2. ddeexec

功　能：向 DDE 服务器应用程序发送用于执行的命令。

语　法：rc = ddeexec (channel, command, item, timeout)

说　明：函数 ddeexec 通过已经建立起的 MATLAB 同服务器应用程序间的对话，向 DDE 服务器程序发出一条用于执行的命令，它的各输入参数的含义如下：

- channel 为前面使用函数 ddeinit 建立的 DDE 对话；
- command 为用户希望执行的命令；

- item 为一个可选的参数，为用于数据传输的 DDE 项，决定了传输数据的格式，对于大多数的应用程序均不使用该参数；
- timeout 同样为一个可选参数，它是一个用于指定超时时间长短的数量，单位为微秒，默认情况下为 3000ms，即三秒。

如果该函数执行成功，其返回值为 1，否则为 0。

举　例：% 通过已经建立的通道 channel 向 Excel 发送一个执行命令

```
rc = ddeexec (channel, ' [formula.goto ("r1c1")]');
```

3. ddepoke

功　能：从 MATLAB 向 DDE 服务器应用程序发送数据。

语　法：rc = ddepoke (channel, item, data, format, timeout)

说　明：通过函数 ddepoke，用户可以使用已经建立起的 MATLAB 同服务器应用程序间的对话，向 DDE 服务器程序发送数据。在发送数据时，函数将按下面的要求对数据格式进行转换：

- 对于字符串矩阵，函数一个元素一个元素地将矩阵转化为字符并存放在一个缓冲之中，然后将缓冲发送到服务器应用程序中；
- 对于数值矩阵，函数将矩阵的内容转组织为用 tab 字符表示列结束和用回车换行表示行结束的数字传送到服务器应用程序中；并且要求只能传送非稀疏矩阵的实数部分的数据。

函数 ddepoke 的各输入参数的含义如下：

- channel 为前面使用函数 ddeinit 建立的 DDE 对话；
- item 为用于数据传输的 DDE 项，决定了传输数据的格式；
- data 为用户希望传送的数据；
- format 为一个可选的参数，是一个数量，用于指定 Windows 剪贴板的数据格式，MATLAB 作为客户端时，只支持文本格式的剪贴板数据，相应的值为 1；
- timeout 同样为一个可选参数，它是一个用于指定超时时间长短的数量，单位为微秒，默认情况下为 3000ms，即三秒。

如果该函数执行成功，其返回值为 1，否则为 0。

举　例：%通过已经建立的通道 channel 向 Excel 发送一个 5×5 的矩阵

```
rc = ddepoke (channel, 'r1c1: r5c5', eye (5));
```

4. ddereq

功　能：从 DDE 服务器应用程序请求数据

语　法：data = ddereq (channel, item, format, timeout)

说　明：通过函数 ddereq，用户可以使用已经建立起的 MATLAB 同服务器应用程序间的对话，向 DDE 服务器程序请求数据。如果函数执行成功，将返回一个包含请求的数据的矩阵；如果函数执行失败，将返回一个空矩阵。函数的各输入参数的含义如下：

- channel 为前面使用函数 ddeinit 建立的 DDE 对话；
- item 为用于数据传输的 DDE 项，决定了传输数据的格式；
- format 为一个可选的参数，是一个数量，用于指定 Windows 剪贴板的数据格式，MATLAB 作为客户端时，只支持文本格式的剪贴板数据，相应的值为 1；
- timeout 同样为一个可选参数，它是一个用于指定超时时间长短的数量，单位为微秒，默认情况下为 3000ms，即三秒。

举　例：%通过已经建立的通道 channel 向 Excel 请求数据

```
mymtx = ddereq (channel, 'r1c1: r10c10');
```

5. ddeterm

功　能：终止 DDE 服务器应用程序与 MATLAB 间的 DDE 对话

语　法：rc = ddeterm (channel)

说　明：通过函数 ddeterm，用户可以终止服务器应用程序与 MATLAB 间的 DDE 对话，该函数仅由一个输入参数，即用于标识已经建立的通道的 channel。

举　例：% 终止 DDE 对话

```
rc = ddeterm (channel);
```

6. ddeadv

功　能：在 MATLAB 同 DDE 服务器应用程序之间建立一个顾问连接

语　法：rc = ddeadv (channel, item, callback, upmtx, format, timeout)

说　明：在通过函数 ddeinit 在 MATLAB 同 DDE 服务器应用程序之间建立起一个 DDE 对话之后，通过函数 ddeadv，用户可以在 MATLAB 同 DDE 服务器应用程序之间针对某些数据，建立起一个顾问连接。通过顾问连接，用户可以在服务器程序的数据发生变化时，通知客户程序即 MATLAB。在 MATLAB 作为客户端程序时，支持两种类型的顾问连接，它们分别称为热连接（hot link）和温连接（warm link），它们之间的区别在于当数据发生改变时，热连接将导致服务器立即向 MATLAB 发送数据；而温连接仅在 MATLAB 请求数据时才通知 MATLAB 数据已经发生了变化。函数 ddeadv 的各输入参数的含义如下：

- channel 为前面使用函数 ddeinit 建立的 DDE 对话；
- item 为用于数据传输的 DDE 项，决定了传输数据的格式；
- callback 为服务器数据发生变化时，服务器通知 MATLAB，MATLAB的响应动作；
- upmtx 为一个可选参数，是一个用来存放更新数据矩阵名字的字符串，当使用该参数时，如果服务器的数据发生变化，将导致 upmtx 表示的被更改的数据所更新。如果在工作空间中存在同名的矩阵，那么该矩阵将被覆盖，如果不存在同名矩阵，将创建一个新的矩阵；
- format 为一个可选的参数，是一个数量，用于指定 Windows 剪贴板

的数据格式，MATLAB 作为客户端时，只支持文本格式的剪贴板数据，相应的值为 1；

- timeout 同样为一个可选参数，它是一个用于指定超时时间长短的数量，单位为微秒，默认情况下为 3000ms，即三秒。

如果该函数执行成功，其返回值为 1，否则为 0。

举　例： 下面是一个由 MATLAB 提供的示例程序，其源代码如下，我们对其加以注释：

```
% 初始化与 Excel 的 DDE 对话
chan = ddeinit ('excel', 'Sheet1');

% 设置输出数据的范围
range = 'r1c1: r20c20';
% 产生一个图形窗口，并得到用于输出的矩阵 z
h = surf (peaks (20));
z = get (h, 'zdata');
% 将矩阵 z 输出到 Excel 的电子表格中
rc = ddepoke (chan, range, z);
% 针对数据 z 在 MATLAB 同 Excel 之间建立一个热连接，
% 同时通过回调函数利用数据 zdata 和 cdata 对曲面 h 重绘
rc = ddeadv (chan, range,'set (h,''zdata'', z); set (h,''cdata'', z);','z');
% 在图形窗口中建立一个按钮，并提供响应函数，用于终止热连接
% 并且关闭图形窗口
c = uicontrol ('String','&Close','Position', [5 5 80 30],...
              'Callback','rc = ddeunadv(chan,range);ddeterm(chan);close;');
```

执行上面的代码后，MATLAB 将显示图 7.11 所示的图形：

这时在 Excel 中对数据进行修改，将会导致该图形的变化。例如将电子表格中的第一个数据改为 10，图形将变为如下形式（见图 7.12），在坐标（0，0，0）处图形有一个明显的突起。

7. ddeunadv

功　能： 释放 MATLAB 同 DDE 服务器应用程序之间建立的顾问连接。

语　法： rc = ddeunadv (channel, item, format, timeout)

说　明： 该函数用于释放 MATLAB 同 DDE 服务器应用程序之间建立的顾问连接，它的各输入参数含义如下：

- channel 为前面使用函数 ddeinit 建立的 DDE 对话；
- item 为用于数据传输的 DDE 项，决定了传输数据的格式；
- format 为一个可选的参数，是一个数量，用于指定 Windows 剪贴板的数据格式，MATLAB 作为客户端时，只支持文本格式的剪贴板数据，相应的值为 1；
- timeout 同样为一个可选参数，它是一个用于指定超时时间长短的数量，单位为微秒，默认情况下为 3000ms，即三秒。

如果该函数执行成功，其返回值为 1，否则为 0。

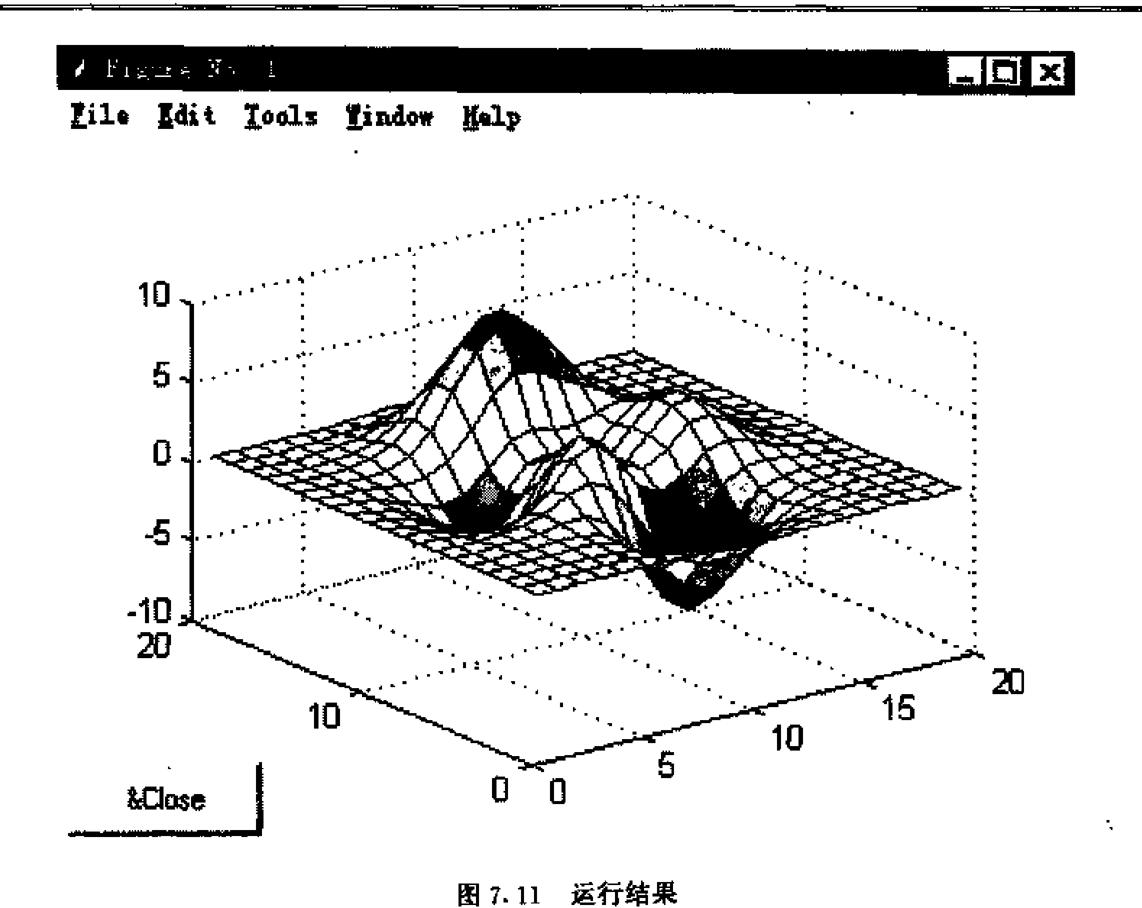


图 7.11　运行结果

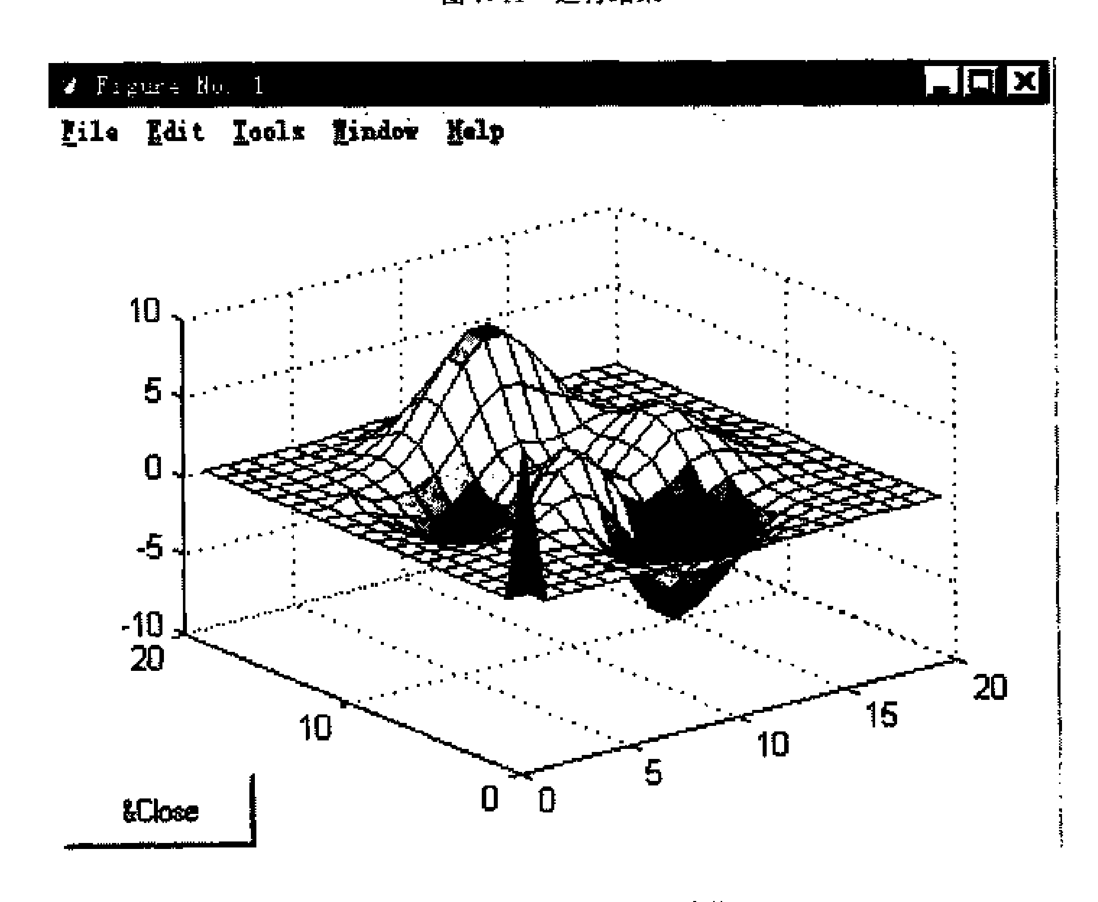


图 7.12　数据变化后的图形

举　例：见函数 ddeadv 的举例。

第8章　MATLAB C++数学函数库的使用

MATLAB C/C++数学函数库是MATLAB为C语言和C++语言提供的两个用于阵列运算的数学函数库，是MATLAB产品家族的重要组成成员。准确地说，MATLAB C/C++数学函数库不是MATLAB应用程序接口的组成部分，而是属于“MATLAB扩展”部分的内容，这从本书的第一章的图1.3可以明显看出。我们之所以将它放在本书中进行讲述的主要原因是：从某种程度上说，MATLAB C/C++数学函数库所提供的功能与MATLAB应用程序接口提供的功能非常类似，即可以充分地利用MATLAB所提供的高效的面向阵列的数值计算能力和大量的内建函数，从而大大地提高算法开发的速度，缩短开发周期。当然MATLAB C/C++数学函数库和MATLAB应用程序接口作为MATLAB产品家族的不同成员，它们之间的差异也是相当巨大的：

首先，基于二者开发的应用程序的目的完全不同。基于MATLAB应用程序接口开发应用程序的目的一般来说可以分为三点：第一，是为了建立MATLAB与其他应用程序间的数据交换，这主要是MAT文件应用程序来完成的；第二，是为了充分利用其他应用程序的优点如计算速度快和已有的算法程序，从而避免重复的开发，这主要是通过MEX文件来完成的；第三，是为了拓广MATLAB的应用范围和应用手段，如在Visual Basic和Visual C++中对MATLAB进行调用，这主要是通过MATLAB引擎和MATLAB ActiveX来完成的。而基于MATLAB C/C++数学函数库开发应用程序的目的相对来说就简单许多了，就是为了利用现有的MATLAB所提供的功能，简化在C语言和C++语言中对矩阵的处理。

其次，基于二者开发的应用程序的执行方式也完全不同。基于MATLAB应用程序接口开发的应用程序主要可以分为三种，即MEX文件、MAT文件应用程序和MATLAB引擎应用程序，其中MEX文件为一种动态链接库程序，它不能脱离MATLAB的工作环境而执行，必须在MATLAB的工作环境内部，通过MATLAB调用才能运行；MAT文件应用程序是一种可以独立执行的应用程序，但它完成的功能非常有限，只能用于数据交换，而不能利用MATLAB所提供的功能来完成计算任务；MATLAB引擎应用程序也是一种可以独立执行的应用程序，但是在应用程序执行时，将在后台启动一个MATLAB进程，用于接收从应用程序发送来的指令并执行，然后按要求返回计算结果。综上所述，可以看出，基于MATLAB应用程序接口开发的应用程序并不是一种独立可执行的应用程序，仍然需要依靠MATLAB。而基于MATLAB C/C++数学函数库开发的应用程序则完全不同，一旦它们建立成功，就无需依靠MATLAB，可以完全独立地执行，与MEX文件和MATLAB引擎应用程序相比，它们主要具有以下几个明显的优点：

- 执行速度快；
- 内存需求小；
- 可以发布给没有MATLAB的用户使用。

但是，也正是由于基于MATLAB C++数学函数库编写的应用程序的独立可执行

性，导致了它们具有以下的缺点：

- 用户不能在基于 MATLAB C++数学函数库的应用程序中使用 MATLAB 的函数 eval () 和 input ()；
- 用户不能在基于 MATLAB C++数学函数库的应用程序中调用 MATLAB 的图形句柄系统的函数；
- 用户不能在基于 MATLAB C++数学函数库的应用程序中调用 MATLAB 工具箱的函数；
- 用户不能在基于 MATLAB C++数学函数库的应用程序中访问 Simulink；
- MATLAB 中的一些语法在 C 语言和 C++语言中得不到支持，例如“:”和“[]”。

由于以上的优缺点，基于 MATLAB C++数学函数库编写的应用程序非常适合应用于需要进行大规模计算并且没有图形输出的场合，当然用户也可以使用第三方的图形系统来显示 MATLAB C++数学函数库的计算结果。

总体说来，基于 MATLAB C++数学函数库和基于 MATLAB 应用程序接口编写的应用程序各有所长，用户可以根据需求进行实际选择。

在本章中，我们将对 MATLAB C++数学函数库进行一些简单的介绍，使读者掌握基本的 MATLAB C++数学函数库的概念和使用方法，能够编写简单的程序，而对一些较为深入的内容不予说明，因为这部分内容并不是本书的重点。同时本书忽略了 MATLAB C数学函数库的讲解，这主要是因为 MATLAB C++数学函数库完全构建在 MATLAB C 数学函数库的上层，通过C++的类的机制，对原有的 MATLAB C 数学函数库的内容进行了封装，使得函数库更加易于使用。

8.1　MATLAB C++数学函数库简介

8.1.1　什么是 MATLAB C++数学函数库

简而言之，MATLAB C++数学函数库就是一个由 MATLAB 提供的基于C++语言的数学函数库，其中包含了大约 400 个 MATLAB 的数学函数，不但包括了大量的 MATLAB 内建数学函数，而且包含了许多在 MATLAB 中被声明为 M 文件的数学函数，为 C++语言中矩阵的计算和处理提供了方便而快捷的手段。MathWorks 公司提供 MATLAB C++数学函数库主要是出于两个方面的目的：

- 对于编写 M 文件的 MATLAB 程序员来说，可以利用已有的编写 M 文件的知识和经验，花费极小的代价，利用该数学函数库来编写代码极为类似于 MATLAB 的 M 文件的，而性能却显著提高的，并且可以脱离 MATLAB 的解释性环境独立执行的应用程序；
- 对于C++的程序员来说，该数学函数库则提供了一个非常自然而又非常牢固的编程接口和大量的功能丰富的矩阵计算和处理函数，可以使C++程序员方便快捷地利用其获得 MATLAB 所提供的强大的矩阵计算和处理能力，从而大大地提高程序设计的效率。此外，该数学函数库还对程序员隐蔽了大量的编程细节，例

如内存管理等，可以使程序员用一种非常简单直接的语法按他们的想法去编程，集中全力去解决问题，而无须考虑工具本身。

8.1.2 类 mwArray

为了方便线性代数算法的开发，在 MATLAB C++数学函数库中，定义了大量相关的类和函数，其中最为重要和最为基础的类就是类 mwArray，它相对应于 MATLAB 中的阵列数据类型，是 MATLAB C++数学函数库中最为基本的数据类型，也是用户学习使用 MATLAB C++数学函数库的关键。类 mwArray 支持大部分的 MATLAB 运算符和所有的 MATLAB 数学函数，不支持的 MATLAB 运算符仅有以下几种：

\, ./, .\, .*, .^

这主要是因为在 C++语法中，它们为非法的标识符，不过在 MATLAB C++数学函数库中提供了相应的函数来替代这些运算符完成相应的功能。这里必须明确指出的一点是，千万不要将类 mwArray 与前面的在 MATLAB 应用程序接口中的 mxArray 结构体相混淆，mxArray 结构体是基于 C 语言定义的相对应于 MATLAB 阵列的数据类型，而类 mwArray 则是利用 C++语言的类机制对结构体 mxArray 和一些相应的函数进行封装后的结果。

在每一个 mwArray 的类对象的内部，均包含了一个指向 MATLAB 阵列的指针，也正是因为这个原因，一个 mwArray 类对象的属性是一个 MATLAB 阵列属性的超集，其中包括了所指向的 MATLAB 阵列的所有属性。在这些属性中，不但包括了阵列的大小、形状，即阵列的维数和各维的大小，而且还包括了一个或两个数组的数据，这主要由阵列的类型来确定。当阵列为复数类型时，其中第一个数组用来存放复数实数部分的数据，而第二个数组则用来存放复数虚数部分的数据；当阵列为实数类型时，属性中仅包含第一个数组用于存放数据，而不包含第二个数组。必须注意的一点是，在两个数组中数据均以列优先的原则存放。

在 MATLAB C++数学函数库 2.0 版本中，对类 mwArray 的功能进行了进一步的扩展，使得类 mwArray 能够对应于更多的 MATLAB 阵列类型，而且还包括了对一些 MATLAB V5.X 才具有的阵列类型提供了支持，例如多维阵列、单元阵列、MATLAB 结构体和稀疏矩阵。不过，在 MATLAB C++数学函数库 2.0 版本中，类 mwArray 并没有对 MATLAB 的整数类型的阵列提供支持，主要包括 int8、int16、int32、unit8、uint16 和 uint32 类型的阵列，同时在 MATLAB C++数学函数库中，也没有提供对 MATLAB 中有关处理整数类型阵列的数学函数的封装，所以用户不能利用 MATLAB C++数学函数库来处理整数类型的阵列，这一点必须引起注意。

总的来说，类 mwArray 是一个极为精简的类，其中仅包含了一些与阵列操作紧密相关的成员函数，如构造函数、析构函数、转换函数、内存管理函数、赋值函数、输入输出函数、索引函数等，而不包括任何的与数学运算相关的函数和运算符。同时类 mwArray 为所有的这些数学函数和运算符提供了一个统一的接口，通过这个接口，外部的数学函数和运算符可以相当方便地对类 mwArray 的对象进行数学计算。这种方法使得类 mwArray 的实现变得相当简单，无论是对于类 mwArray 的开发者还是对于类 mwArray 的使用者来说，都可以从这种方法获得极大的利益。首先对于开发者来说，只要在不修

改接口的前提下，可以任意对类 mwArray 的内容进行修改，而无须对外部的数学函数和运算符进行修改，为以后的升级提供了方便；而对于使用者来说，类 mwArray 的实现越简单，越有利于用户的学习和使用。除此之外，类 mwArray 的这种实现方法，还可以避免由于 MATLAB 函数的多种形式调用和多输出而造成的混淆和冲突。

有关类 mwArray 中各成员函数的定义，读者可以参见目录

MATLAB 根目录\EXTERN\INCLUDE\CPP

下的头文件 dblmtrs.h，对类 mwArray 进行深入的理解，非常有利于对 MATLAB C++ 数学函数库的学习。

下面我们将对类 mwArray 的成员函数分类进行讲解。

1. 构造函数

在类mwArray 中，共提供了 18 个不同的构造函数，它们的定义分别如下：

```
mwArray ();
mwArray (const char *str);
mwArray (mwString str);
mwArray (Dimension, Dimension, mwNumericInitAction);
mwArray (Dimension, Dimension, double *real, double *imag = 0,
        int copy = 1);
mwArray (Dimension, Dimension, int *real, int *imag = 0);
mwArray (Dimension, Dimension, unsigned short *real,
        unsigned short *imag = 0);
mwArray (const mwArray &mtrx);
mwArray (const MatlabMatrix *mtrx, mwBool freeflg =1);
mwArray (const MatlabMatrix *mtrx, mwBool freeflg, mwBool staticflg);
mwArray (double start, double step, double stop);
mwArray (const mwArrayIndex &idx);
mwArray (const mwNumericSubArray & a);
mwArray (MatrixRef *matref);
mwArray (double);
mwArray (double, double);
mwArray (int);
mwArray (int, const char * * );
```

其中 Dimension 和 mwNumericInitAction 的定义分别如下：

```
typedef int Dimension;
enum mwNumericInitAction
{
    mwEyeAct,       // Places ones on the main diagonal
    mwOnesAct,      // Fill array with ones
    mwZerosAct,     // Fill array with zeros
    mwRandAct,      // Fill array with random numbers
    mwRandnAct,     // Fill array with normally distributed random numbers
```

```
    mwMagicAct        // Make a magic square (only works for matrices) .
};
```

通过以上的这些构造函数，用户可以使用各种不同的方式，构造各种不同数据类型的、与MATLAB中各种不同类型阵列相对应的mwArray类对象。下面我们对其中一些较为常用的构造函数进行说明。

- mwArray ();

  该函数为类mwArray的默认构造函数，通过它，用户可以创建一个未初始化的mwArray类对象（或称为阵列），例如

  ```
  mwArray A;
  ```

  当将该对象作为参数传递给MATLAB C++数学函数库中的函数时，将产生一个警告信息，因为在对象中不包含任何的信息，所以在将由此构造函数创建的类对象传送给MATLAB C++数学函数库中的函数之前，必须进行赋值。请读者注意一点，这里我们将mwArray类对象称为了阵列，主要是出于描述方便和与MATLAB中概念统一的目的，在后续的内容中，我们都将采用这种说明方法；

- mwArray (const char *str);

  通过该函数，用户可以构造一个字符串类型的阵列，并且通过输入的字符串指针指向的字符串进行初始化，例如

  ```
  mwArray A ("MATLAB Rules");
  ```

- mwArray (int32, int32, double *, double *);

  通过该函数，用户可以创建一个指定行数、列数以及虚部和实部数据的二维复数阵列，例如

  ```
  double real [] = { 1, 2, 3, 4 };
  double imag [] = { 5, 6, 7, 8 };
  mwArray A (2, 2, real, imag);
  ```

  这里必须明确的一点是，数组中实部和虚部的数据必须以列优先的原则存放；此外函数声明中的最后一个参数为一个可选参数，当用户没有给出该参数时，函数将创建一个实数类型的阵列；

- mwArray (const mxArray *);

  通过该函数，用户可以通过一个现成的mxArray结构体对象来创建一个阵列，该mxArray结构体对象既可以为MATLABC数学函数库中函数的返回结果，也可以为MATLAB应用程序接口中函数的返回结果，例如

  ```
  mxArray *m = mlfScalar (1);
  mwArray mat (m);
  ```

- mwArray (const mwArray&);

  通过该函数，用户可以通过一个输入的阵列来构造一个新的阵列，并且将输入阵列的内容复制到新创建的阵列中，例如

  ```
  mwArray A = rand (4);
  mwArray B (A);
  ```

实际上，该函数并没有将对输入阵列的内容进行复制，而只是通过一个指针进行引用，直至数据被修改时，才进行真正的拷贝；

- mwArray (double, double, double);

通过该函数，用户可以创建一个元素间等差值的阵列，例如用户希望创建一个形如[1.0，1.5，2.0，2.5，3.0，3.5]的阵列，则可以通过下面的语句来完成：

```
mwArray A (1.0, 0.5, 3.5);
```

- mwArray (int32, int32, int32);

通过该函数，用户可以创建一个元素间等差值且元素为整型的阵列，例如用户希望创建一个形如 [1，5，9，13] 的阵列，则可以通过下面的语句来完成：

```
mwArray (1, 4, 13);
```

- mwArray (const mwSubArray&);

通过该函数，用户可以由输入的 mwSubArray 类对象来构造一个新的阵列，例如

```
mwArray A = rand (4);
mwArray B (A (3, 3));
```

- mwArray (double);

通过该函数，用户创建一个 1×1 的双精度浮点类型的阵列，例如

```
mwArray A (17.5);
```

- mwArray (int);

通过该函数，用户可以从一个输入的整数创建一个 1×1 的阵列，例如

```
mwArray A (17);
```

2. 索引和下标函数

类 mwArray 中，提供了三个关于对阵列（即 mwArray 类对象）元素进行索引和下标操作的函数和运算符，分别为 operator ()，cell () 和 field ()，它们的功能各不相同：

- operator ()

operator () 是类 mwArray 定义的用来对多维阵列的元素进行索引的运算符，同时还可以用来对由结构体阵列和单元阵列的元素进行索引。在 MATLAB C++数学函数库中，对阵列的索引操作通过三个类，即类 mwArray、类 mwNumericSubArray 和类 mwIndex 的交互来完成。

在类 mwArray 的声明中，包含了一系列经过重载的 operator () 运算符，允许用户对从一维到 32 维的阵列进行索引操作，它们的定义如下：

```
/* 1维 */
mwArray operator () (const mwVarargin &a) const;
mwNumericSubArray operator () (const mwVarargin &a);
/* 2维 */
mwArray operator () (const mwArray &a1, const mwArray &a2) const;
mwNumericSubArray operator () (const mwArray &a1, const mwArray &a2);
```

```
……
/* 32维 */
mwArray operator () (const mwArray &a1, const mwArray &a2,
                     const mwArray &a3, const mwArray &a4,
                     ……
                     const mwArray &a32) const;
mwNumericSubArray operator () (const mwArray &a1, const mwArray &a2,
                               const mwArray &a3, const mwArray &a4,
                               ……
                               const mwArray &a32);
```

其中在每一维的索引操作中均包含一对 operator ()。

- cell ()

  cell () 是类 mwArray 定义的用来对单元的内容进行索引操作的函数，在类 mwArray 的声明中，它们的定义如下：

```
mwArray cell (const mwVarargin &RI1, const mwArray &OI2=mwArray:: DIN,
              const mwArray &OI3=mwArray:: DIN,
              const mwArray &OI4=mwArray:: DIN,
              ……
              const mwArray &OI32=mwArray:: DIN ) const;
mwNumericSubArray cell (const mwVarargin &RI1,
                        const mwArray &OI2=mwArray:: DIN,
                        const mwArray &OI3=mwArray:: DIN,
                        ……
                        const mwArray &OI32=mwArray:: DIN );
```

函数允许用户向其传递 32 个参数，当用户希望对超过 32 维的单元进行索引时，则用户必须构造一个类 mwVarargin 的对象，并将其作为第一个参数传送给函数 cell ()。

- field ()

  field () 是类 mwArray 定义的用来对结构体的域进行索引操作的函数，在类 mwArray 的声明中，它们的定义如下：

```
mwArray field (const char * fieldname) const;
mwNumericSubArray field (const char * fieldname);
```

3. 转换函数

在类mwArray 的声明中，仅包含一个转换函数，其定义如下：

```
operator double () const;
```

通过该函数，用户可以将一个 1×1 的非复数的数值阵列，转换为一个双精度浮点类型的实数。

4. 内存管理函数

由于 MATLAB C++数学函数库拥有其自身的内存管理机制，因此运算符 new 和

运算符 delete 在类 mwArray 中被进行了重载，它们的定义和实现分别如下：

```
inline void * mwArray:: operator new (size_t size)
  {
    assert (size == sizeof (mwArray));
    return mxMalloc (size);
  }
inline void mwArray:: operator delete (void * ptr)
  {
    mxFree (ptr);
  }
```

5. 输入和输出函数

为了完成对 mwArray 类对象即阵列的输入和输出任务，类 mwAray 对输出运算符“<<”和输入运算符“>>”进行了重载，它们的声明和定义分别如下：

```
friend inline ostream& operator<< (ostream &os, const mwArray&);
  inline ostream& operator<< (ostream &os, const mwArray &m)
  {
    m.Write (os);
    return os;
  }
  inline ostream& operator<< (ostream &os, const mwNumericSubArray &a)
  {
    os << (mwArray) a;
    return os;
  }

  friend inline istream& operator>> (istream &is, mwArray&);
  inline istream& operator>> (istream &is, mwArray &m)
  {
    m.Read (is);
    return is;
  }
```

通过调用运算符<<，用户可以将一个阵列（即 mwArray 类对象）插入到一个指定的流中，如果该指定的流为 cout，那么阵列的内容将被输出到终端屏幕上。如果用户通过命令行，对标准的输出流进行了重定向，那么阵列的内容将被输出到用户指定的位置。通过调用运算符>>，用户可以从一个指定的流中读入阵列，该流可以为任何的C++流对象，例如标准终端输入 cin、文件或一个字符串。

从上面运算符<<和运算符>>的声明中，可以看到，实际上，这两个运算符并不是类 mwArray 的真正的成员函数，而是类 mwArray 的友员函数，这一点必须注意。此外在运算符<<和运算符>>的实现中，我们可以看到，它们分别调用了函数 Write () 和 Read ()，这两个函数为类 mwArray 的公用成员函数，它们的定义分别如下：

```
void Read (istream&, int is2D = 0);
void Write (ostream&) const;
```

函数 Read () 用来从一个流中读入阵列，输入阵列的格式必须为如下形式：

$$\alpha * [a, b, \cdots, c; \cdots; \cdots]$$

其中 α 为系数，* 为乘号，符号 [ ] 内为阵列元素，a、b 和 c 为单个的阵列元素，阵列中每一行的元素利用分号进行分隔，而利用逗号分隔单个的元素。

函数 Write () 用来将阵列格式化后输出到指定的流中。

6. 大小函数

在类 mwArray 中，提供了三个不同的获取大小的函数，分别完成不同的功能，它们的定义和实现分别如下：

```
int32 Size () const;
int32 Size (int32 dim) const;
int32 Size (int32 * dims, int maxdims=2) const;

inline int32 mwArray:: Size () const
{
    return mxGetNumberOfDimensions (GetData ());
}

inline int32 mwArray:: Size (int32 dim) const
{
    MatlabMatrix *matrix = GetData ();
    if (matrix ! = 0)
    {
        if (dim > Size ())
        {
            return 1;
        }
        const int *dimarray = mxGetDimensions (GetData ());
        return dimarray [dim-1];
    }
    return 0;
}

inline int32 mwArray:: Size (int32 * dims, int maxdims=2) const
{
    int i, size = Size ();
    const int * dimarray = mxGetDimensions (GetData ());
    int ndims = size>maxdims? maxdims: size;
    for (i = 0; i < ndims; i++)
    {
        dims [i] = (int32) dimarray [i];
```

```
    }

    if (size > maxdims)
    {
        int last = maxdims－1;
        for (; i < size; i++)
        {
            dims [last] *= (int32) dimarray [i];
        }
    }
    else if (maxdims > size)
    {
        for (; i < maxdims; i++)
        {
            dims [i] = 1;
        }
    }
    return size;
}
```

它们的函数名完全相同，而输入参数却各不相同，用于不同情况，这是利用了C++的函数重载机制。其中第一个函数可以用来获取阵列的维数，第二个函数可以用来获取指定维数的大小，而第三个函数则可以同时获得多个维的大小，并存储在一个数组之中，其中数组 dims 的元素个数不得少于输入参数 maxdims。如果输入参数 maxdims 的值小于阵列的维数，则数组 dims 中的最后一个元素的值为所有余下维数的大小的乘积，函数的返回值为阵列的总的维数。

7. *数据提取函数*

在类 mwArray 中，提供了若干个成员函数，通过这些成员函数，用户可以方便地获取 mwArray 类对象（即阵列）中的数据。分别说明如下：

- GetData ()

  函数GetData () 的定义如下：

  ```
  MatlabMatrix *GetData () const;
  ```

其返回值为一个 mxArray 结构体类型的指针。通过该指针和 MATLAB 应用程序接口中的 mx－函数，就可以对 mwArray 类对象中的数据进行访问了，例如

```
mwArray A = magic (17);
double *real_data = mxGetPr (A.GetData ());
```

这里指针 real_data 维实际指向 mwArray 类对象的指针，所以不能对该指针进行释放操作，否则将导致内存错误。

在类 mwArray 中，与函数 GetData () 功能相对应的函数为 SetData ()，通过该函数，用户可以对类 mwArray 的对象的数据进行设置。不过函数 SetData () 的使用必须非常小心，因为它可以骗过 MATLAB C++数学函数库的自动内存管理

机制，导致内存的泄漏，建议读者尽量不要使用该函数进行mwArray类对象的数据设置操作，而最好通过类mwArray的构造函数来完成。

· ExtractScalar（）和ExtractData（）

函数ExtractScalar（）和函数ExtractData（）为用户提供了一种更加方便而且更加安全的获取mwArray类对象的方法。函数ExtractScalar（）的定义如下：

```
double ExtractScalar (double&, int32) const;
double ExtractScalar (int32) const;
```

通过这两种版本的函数，用户可以从实数或复数类型的mwArray类对象中轻松地获得单个的数据。函数ExtractScalar（）在执行时，将一个m×n的阵列视为一个1×（m＊n）的阵列。

函数ExtractData（）的定义如下：

```
void ExtractData (int32 * );
void ExtractData (double * );
void ExtractData (double * , double * );
```

通过这三种版本的函数，用户可以将指定的mwArray类对象的数据拷贝到指定的C++数组中。下面为一个简单的例子：

```
mwArray A = magic (11) + (rand (11) * i ());
double rdata, cdata;
rdata = A.ExtractScalar (9);
rdata = A.ExtractScalar (cdata, 17);
int32 *integers = new int32 [ 11 * 11 ];
A.ExtractData (integers);
double *real_data = new double [ 11 * 11 ];
double *complex_data = new double [ 11 * 11 ];
A.ExtractData (real_data);
A.ExtractData (real_data, complex_data);
```

· ToString ()

函数ToString ()在类mwArray中的定义为

```
mwString ToString () const;
```

通过该函数用户可以方便地从一个字符串类型的阵列中获取数据，其返回值为一个类mwString的类对象。例如

```
mwArray A = "abcdefg";
mwString s = A.ToString ();
char *c = strdup ( (char * ) s);
```

8. 其他成员函数

除了上面的一些成员函数外，在类mwArray中，还提供了下面的一些成员函数（部分）：

```
mwBool IsDIN () const { return (data == mwArray:: DIN.data); }
mwBool IsEmpty () const;
mwBool IsNullMatrix () const;
mwBool IsString () const;
mwBool IsInitialized () const;
mwBool IsUnassigned () const;
int32 EltCount () const;
static void SetLineWidth (int32 i) { line_width = i; }
static int32 GetLineWidth () { return line_width; }
void Unshare ();
void Uninitialize ();
void CopyInputArg (const mwArray &a);
void CopyOutputArg (const mwArray &a);
void MakeDIN ();
static int Constructors ()     { return ConstructorCount; }
static int Destructors ()      { return DestructorCount; }
static int ScalarCtors ()      { return ScalarConstructor; }
```

它们全部在头文件 dblmtrs.h 中得到声明，有兴趣的读者可以自行参阅该文件。

8.1.3 基于 MATLAB C++数学函数库应用程序的建立

在使用 MATLAB C++数学函数库建立可独立执行的应用程序之前，用户必须对自己所使用的C++编译器和链接器进行正确的配置，否则将无法得到正确的可执行的应用程序。然而，对于习惯了使用集成工作环境的用户来说，这却是一个非常麻烦的工作，正因为如此，Mathworks 公司为了方便用户，提供了帮助用户进行C++编译器和链接器设置的命令行工具 mbuild，通过命令，用户可以在系统提示下，一步一步地完成配置工作，并生成一个默认的选项文件，记录配置信息。在以后不对选项文件进行指定的情况下，命令 mbuild 将调用该默认的选项文件对基于 MATLAB C++数学函数库的应用程序源程序进行编译链接。

在本小节中，我们将基于 Microsoft Windows NT 4.0 操作系统和 Visual C++编译链接器，首先对基于命令行工具 mbuild 的 MATLAB C++数学函数库应用程序的建立进行说明，然后对 MATLAB 所提供的一些选项文件进行说明。

1. 命令行工具 mbuild

命令行工具 mbuild 是 MathWorks 公司为了方便用户而提供的一个功能强大的工具，用户不但可以通过它对基于 MATLAB C++数学函数库的应用程序的源程序进行编译链接，而且可以帮助用户对 C++编译器和链接器进行设置。mbuild 命令的使用极为简单，如果需要对某个源程序进行编译链接，只需输入以下命令即可：

```
mbuild filename.cpp
```

其中 filename.cpp 为用户希望建立的应用程序的源程序。如果用户在此之前没有对系统进行过C++编译器和链接器的配置，那么 mbuild 命令将显示如下内容：

```
mbuild has detected the following compilers on your machine:
```

```
[1] : Borland compiler in T: \Borland\BC.500
[2] : WATCOM compiler in T: \watcom\c.106

[0] : None
Please select a compiler. This compiler will become the default:
```

提示用户选择 C++编译器，然后进一步要求用户输入 C++编译器信息，其中 [1] 和 [2] 为在用户系统中可以找到的 C++编译器。建议用户最好不要采用这种方式，而是要在对基于 MATLAB C++数学函数库的应用程序的源程序进行编译之前，首先使用 mbuild 命令的 setup 参数，对用于编译和链接源程序的 C++编译链接器进行设置。整个过程如下，首先在 MATLAB 命令提示符下键入以下命令：

```
? mbuild -setup
```

这时系统将启动一个 DOS 命令框，并显示如下内容：

```
Please choose your compiler for building standalone MATLAB applications.

Would you like mbuild to locate installed compilers [y] /n?
```

提示用户是否需要命令 mbuild 自动地对系统中的 C++编译和链接器进行定位，如果选择 y，命令将显示如下内容：

```
Please verify your choices:

Compiler: Microsoft 6.0
Location: e: \Program Files\Microsoft Visual Studio

Are these correct? ( [y] /n):
```

告知用户所找到的 C++编译器和路径，并提示用户确认是否正确，如果选择 n，命令将显示

```
MBUILD.BAT: No compiler was set
```

并终止命令的运行；如果选择 y，命令将以上面的路径信息对编译器进行一定的设置，如果失败，将给出错误信息：

```
MBUILD.BAT: Error: mbuild requires that the Microsoft Visual C++ 6.0
directories " VC98" and " Common" be located within the same parent directory.
(Could not find e: \Program Files\Microsoft Visual Studio\Common. )
```

说明路径错误，并终止命令的运行。这时用户重新运行命令并在第一步中选择 n，告知命令不进行自动编译器定位，接下来命令将显示如下内容：

```
Choose your C/C++ compiler:
  [1] Borland C/C++               (version 5.0, 5.2, or 5.3)
  [2] Microsoft Visual C/C++      (version 4.2, 5.0, or 6.0)
  [3] Watcom C/C++                (version 10.6 or 11)

  [0] None
```

```
Compiler: 2
```

提示用户选择 MATLAB 所提供支持的 C++编译器，选择 2 后命令将显示

```
Choose the version of your C/C++ compiler:
  [1] Microsoft Visual C/C++ 4.2
  [2] Microsoft Visual C/C++ 5.0
  [3] Microsoft Visual C/C++ 6.0

version: 3
```

提示用户选择编译器的版本，选择 3 后，命令将要求用户进一步确认编译器的路径信息

```
Your machine has a Microsoft Visual C/C++ compiler located at e: \Program Files\M
icrosoft Visual Studio.
Do you want to use this compiler [y] /n?
```

如果该路径正确，用户可以直接选择 y，命令将要求用户确认配置的正确性：

```
Are these correct? ( [y] /n): y
```

选择 y，命令将显示

```
The default options file:
" C: \WINNT\Profiles\Administrator\Application
                              Data\MathWorks\MATLAB\compopts.bat"

is being updated...
```

告知用户默认的选项文件已被更新。

如果在确认路径信息时，用户选择了 n，命令将要求用户输入正确的路径：

```
Please enter the location of your C/C++ compiler: [e: \Program Files\Microsoft Vi
sual Studio] e: \Program Files\Microsoft Visual Studio\Vc98
```

当输入路径并回车后，命令同样将要求用户进一步确认路径信息的正确性：

```
Are these correct? ( [y] /n): y
```

选择 y，命令将显示

```
The default options file:
" C: \WINNT\Profiles\Administrator\Application
                              Data\MathWorks\MATLAB\compopts.bat"

is being updated...
```

告知用户默认的选项文件已被更新。

在配置完成之后，必须对配置的正确性进行测试，用户可以在 MATLAB 命令提示符下键入以下命令

```
? mbuild ex1.cpp
```

并回车，这时命令 mbuild 将调用前面所产生的默认的选项对源程序 ex1.cpp 进行编译和链接，如果配置正确，命令在生成可执行程序 ex1.exe 后，将直接返回 MATLAB 命令提

示符下显示信息；如果配置不正确，将显示错误信息。ex1.cpp 为 MATLAB 提供的一个范例程序，位于目录

MATLAB 根目录\EXTERN\EXAMPLES\CPPMATH

中。

通过以上的配置和验证后，用户就可以对基于 MATLAB C++数学函数库的应用程序的源程序进行编译链接生成可执行程序了。

如果在生成默认的选项文件后，再次运行命令

? mbuild-setup

对 C++编译器进行配置，命令 mbuild 将使用新生成的选项文件覆盖旧的选项文件。通过这种方法，用户可以对默认的选项文件进行更新。

mbuild 实际上是一个功能非常丰富的命令行工具，通过它的各种命令参数，允许用户自定义地对程序的源代码进行编译和链接。命令 mbuild 的使用格式如下：

```
mbuild [-c -Dname -f file -g -h [elp] -Ipathname -lang language
                          -link target -setup -output resultname -outdir dirname -O
                          -Uname -v ] sourcefile... [ <objectfile>.obj ... ]
                          [ <libraryfile>.lib ... ] [ <exportfile>.exports ... ]
```

虽然对于大多数的用户来说，命令中的大多数参数是不必要的，仅仅使用最简单的形式

mbuild filename

就可以完成大部分的任务，但是对命令 mbuild 进行深入的理解，将有助于用户更加灵活地掌握命令 mbuild 的使用，因此我们仍将给出命令 mbuild 的所有命令参数，并加以说明，详见表 8.1。

表 8.1　mbuild 命令参数

命令参数	说明
-c	告知命令 mbuild 仅编译而不链接
-D<name>	定义 C++预处理的宏<name>
-f <file>	告知命令 mbuild 使用文件<file>作为选项文件
-g	在编译链接可执行程序时，包含入调试信息
-h [elp]	显示帮助信息
-I<pathname>	将路径名<pathname>包含入命令的文件搜索路径中
-lang <language>	强制语言类型说明，当<language>为 c 或 C 时，表示使用的语言是 C 语言，而当<language>为 cpp 或 C++时，表示使用的语言为 C++语言，该参数主要使用在输入的文件名的后缀不为命令 mbuild 支持时，例如输入文件名为 filename.a
-link <target>	确定输出文件的类型，当<target>为 exe 时，表示建立可执行文件；而<target>为 dll 时，表示建立共享文件，其中 exe 为默认选项
-outdir <dirname>	指定存放建立的可执行程序的路径<dirname>；如果在使用参数-output 时，给出了全路径名，则不能使用该参数
-output <name>	建立一个名字为<name>的可执行程序，后缀名由命令自动添加
-O	建立一个优化的可执行程序
-setup	设置默认的选项文件
-U<name>	取消 C++预处理宏<name>的定义
-v	列出所使用的 C++编译器和链接器的所有设置选项

2. 选项文件

MATLAB V5.3 版本为不同的 C++编译器提供了不同的选项文件，在 Widows 操作系统中，这些选项文件均放在目录

MATLAB 根目录\bin

中，详见表 8.2。

表 8.2　选项文件

类　　型	版　本	选项文件名
Microsoft Visual C++	4.2	msvccompp.bat
	5.0	msvc50compp.bat
	6.0	msvc60compp.bat
Boland C++	5.0	bcccompp.bat
	5.2	bcc52compp.bat
	5.3	bcc53compp.bat
Watcom C++	10.6	watccompp.bat
	11	wat11ccompp.bat

通过在命令 mbuild 中使用参数-f 指定这些选项文件中的某一个，用户可以轻松地使用各种类型的编译器建立基于 MATLAB C++数学函数库的应用程序，而无须频繁地使用命令 mbuild －setup 对默认的选项文件进行修改。

使用命令 mbuild －setup 产生的默认的选项文件名为 compopts.bat，存放于路径

c：\winnt\profiles\administrator\application data\mathworks\matlab

下。该路径对于不同的操作系统，或同一操作系统的不同用户可能不同。下面我们对该文件进行一定的分析，以加深读者对应用程序建立过程的理解。

选项文件 compopts.bat 的内容如下：

```
@echo off
rem compopts.bat

rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rem (1) 普通参数设置
rem * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

set MATLAB=%MATLAB%
set MSVCDir=e：\Program Files\Microsoft Visual Studio\Vc98\
set MSDevDir=%MSVCDir%\..\Common\msdev98
set PATH=%MSVCDir%\BIN;%MSDevDir%\bin;%MATLAB_BIN%;%PATH%
set INCLUDE=%MSVCDir%\INCLUDE;%MSVCDir%\MFC\INCLUDE;
            %MSVCDir%\ATL\INCLUDE;%INCLUDE%
set LIB=%MSVCDir%\LIB;%MSVCDir%\MFC\LIB;%LIB%
```

```
rem ****************************************
rem (2) 编译器参数设置
rem ****************************************

set COMPILER=cl
set OPTIMFLAGS=-O2
set DEBUGFLAGS=-Zi
set CPPOPTIMFLAGS=-O2
set CPPDEBUGFLAGS=-Zi
set COMPFLAGS=-c -Zp8 -G5 -W3 -nologo
set CPPCOMPFLAGS=-c -Zp8 -G5 -W3 -nologo -Zm500 -GX -MD
                 -I%MATLAB%\extern\include\cpp -DMSVC -DIBMPC -DMSWIND
set DLLCOMPFLAGS=-c -Zp8 -G5 -W3 -nologo -DMSVC -DIBMPC -DMSWIND
set NAME_OBJECT=/Fo

rem ****************************************
rem (3) 库创建命令 (预编译过程)
rem ****************************************

set PRELINK_CMDS1=lib /def:"%MATLAB%\extern\include\libmmfile.def"
                  /machine: ix86 /OUT:%LIB_NAME%1.lib /NOLOGO
set PRELINK_CMDS2=lib /def:"%MATLAB%\extern\include\libmcc.def" /machine: ix86
                  /OUT:%LIB_NAME%2.lib /NOLOGO
set PRELINK_CMDS3=lib /def:"%MATLAB%\extern\include\libmatlb.def" /machine: ix86
                      /OUT:%LIB_NAME%3.lib /NOLOGO
set PRELINK_CMDS4=lib /def:"%MATLAB%\extern\include\libmx.def" /machine: ix86
                  /OUT:%LIB_NAME%4.lib /NOLOGO
set PRELINK_CMDS5=lib /def:"%MATLAB%\extern\include\libmat.def" /machine: ix86
                  /OUT:%LIB_NAME%5.lib /NOLOGO
rem HGLIB set PRELINK_CMDS6=lib /def:"%MATLAB%\extern\include\libhg.def"
                            /machine: ix86 /OUT:%LIB_NAME%6.lib /NOLOGO
set PRELINK_DLLS=lib /def:"%MATLAB%\extern\include\%DLL_NAME%.def"
                 /machine: ix86 /OUT:%DLL_NAME%.lib /NOLOGO
set DLL_MAKEDEF=type %LIB_NAME%.exports | perl -e " print \" LIBRARY
                %MEX_NAME%.dll\nEXPORTS\n\"; while (<>) {print;}"
                > %LIB_NAME%.def
rem ****************************************
rem (4) 链接器参数设置
rem MATLAB_EXTLIB is set automatically by mex.bat
rem ****************************************
set LINKER=link
set LINKFLAGS=kernel32.lib user32.lib gdi32.lib %LIB_NAME%1.lib
        %LIB_NAME%2.lib %LIB_NAME%3.lib /implib:%LIB_NAME%.lib /nologo
set LINKFLAGS=%LINKFLAGS% %LIB_NAME%4.lib %LIB_NAME%5.lib
```

```
rem HGLIB set LINKFLAGS=%LINKFLAGS% %LIB_NAME%6.lib

set CPPLINKFLAGS=%LINKFLAGS% %MATLAB_EXTLIB%\libmatpm.lib
set DLLLINKFLAGS=%LINKFLAGS% /dll /implib:"%OUTDIR%%MEX_NAME%.lib"
                 /def:%LIB_NAME%.def
set LINKOPTIMFLAGS=
set LINKDEBUGFLAGS=/debug
set LINK_FILE=
set LINK_LIB=
set NAME_OUTPUT=" /out:%OUTDIR%%MEX_NAME%.exe"
set DLL_NAME_OUTPUT=" /out:%OUTDIR%%MEX_NAME%.dll"
set RSP_FILE_INDICATOR=@

rem ****************************************
rem 资源编译器参数设置
rem ****************************************

set RC_COMPILER=rc /fo "%OUTDIR%mexversion.res"
set RC_LINKER=
```

由上面的源代码可以看出，选项文件 compopts.bat 的结构非常清晰，总共可以分为五个部分：

第一部分为普通参数设置，定义了一些关于 MATLAB 以及 C++语言编译器的路径信息，其中%MATLAB%和%MSVCDir%分别代表了 MATLAB 和 Microsoft Visual C++ 6.0 所在的安装路径，其余的一些相关定义，如头文件包含路径和库文件包含路径，都是建立在它们的基础之上；

第二部分为编译器参数设置，在该选项文件中，通过环境变量 COMPILER 设置了编译器为 cl，并且通过环境变量 COMPFLAGS、OPTIMFLAGS、CPPOPTIMFLAGS、CPPDEBUGFLAGS、CPPCOMPFLAGS、DEBUGFLAGS 和 DLLCOMPFLAGS 对编译器进行了全面的设置，其中各参数的含义如下：

-G5 代表针对奔腾处理器进行优化处理
-W3 代表设置警告级别为 3 级
-Zi 代表使能调试信息
-O2 代表以最大速度优化
-nologo 代表禁止版权信息
-c 代表告诉编译器，只对源文件进行编译而不用链接
-Zp8 代表按 8 个字节对准
-MD 代表与库文件 MSVCRT.LIB 链接
MSVC、IBMPC 和 MSWIND 为预定义的宏名

第三部分为库创建命令，即预编译过程，用于创建输入和输出的函数库；

第四部分为链接参数设置，在这部分内容中，首先通过环境变量 LINKER 设置了链接器的名字为 link，然后通过环境变量 LINKFLAGS 设置了链接器的一些参数选项，声

明了一些在生成应用程序时所需要嵌入的库文件，包括在第三部分中生成的库文件；

第五部分为资源编译器参数设置。

8.2 阵列对象的创建和索引

阵列对象，即 mwArray 类对象，是 MATLAB C++数学函数库中最基本的数据类型，几乎所有的库函数均以它们作为计算和处理的对象，学会对它们的操作，对于学习 MATLAB C++数学函数库的使用是至关重要的。在上一节中，我们已经对类 mwArray 的概念及其构成进行了一定的说明，在本节中，我们将详细说明如何使用类 mwArray 来创建各种不同类型的阵列对象，包括数值阵列、单元阵列、结构体阵列字符阵列和稀疏矩阵，它们分别对应于 MATLAB 中的一种阵列类型，同时讲述如何对阵列对象的元素内容进行访问、赋值和修改等操作，即阵列对象的索引操作。

8.2.1 阵列对象的创建

在 MATLAB C++数学函数库中，通过类 mwArray 对五种类型的 MATLAB 阵列类型提供了支持，它们分别为数值阵列、稀疏矩阵、字符阵列、单元阵列和结构体阵列，本小节将对这五种阵列对象的创建进行介绍。有关这五种 MATLAB 阵列的概念在本书的第一章有详细的介绍，这里不再说明。

1. 数值阵列对象的创建

在 MATLAB C++数学函数库中，提供了大量的用于创建数值阵列对象的函数，分别如下：

```
mwArray A;
mwArray (double);
mwArray (int);
mwArray (int, int, double * , double * );
mwArray (int, int, int * , int * );
mwArray (int, int, unsigned short * , unsigned short * );
mwArray ( mxArray * );
mwArray (const mwArray&);
mwArray (int, int, int);
mwArray (const mwSubArray&);
horzcat ();
vertcat ();
cat ();
randn ();
zeros ();
rand ();
empty ();
ones ();
eye ();
```

```
magic ();
```

总的来说，可以将这些函数可以分为五类，如下：

- 使用类 mwArray 的构造函数来创建数值阵列对象，主要包括以下的一些构造函数：

```
mwArray A;
mwArray (double);
mwArray (int);
mwArray (int, int, double * , double * );
mwArray (int, int, int * , int * );
mwArray (int, int, unsigned short * , unsigned short * );
mwArray ( mxArray * );
mwArray (const mwArray&);
mwArray (int, int, int);
mwArray (const mwSubArray&);
```

通过这些构造函数，用户既可以使用默认的构造函数 mwArray A 创建一个未初始化的数值阵列对象，也可以使用函数 mwArray (int, int, double * , double *) 创建指定行数和列数以及实部和虚部数据的数值阵列对象，此外还可以通过 mwArray (int) 和 mwArray (double) 使用输入的单个的双精度实数和整数来创建 1×1 的数值阵列对象，而且还可以通过现有的 mwArray 类对象和 mxArray 结构体对象使用函数 mwArray (const mwArray&) 和 mwArray (mxArray *) 来创建数值阵列对象。在实际操作中，用户可以灵活选取不同的构造函数，下面是一个简单的例子：

```
double data [] = {1, 4, 2, 5, 3, 6};
mwArray C (2, 3, data);
mwArray D (C);
cout << " C =" << C << endl;
cout << " copy of C =" << D << endl;
```

执行这段代码，将显示如下结果：

```
C = [
1 2 3;
4 5 6
]
Copy of C = [
1 2 3;
4 5 6
]
```

这里必须注意的是，阵列对象的数据为列优先存放的。

- 使用阵列创建函数来创建数值阵列对象，主要包含以下一些函数：

```
ones ();
zeros ();
```

```
rand ();
randn ();
empty ();
magic ();
eye ();
```

通过函数 ones ()，用户可以创建一个所有阵列元素为 1 的数值阵列对象；通过函数 zeros()，用户可以创建一个所有阵列元素为 0 的数值阵列对象；通过函数 empty ()，用户可以创建一个相当于 MATLAB 中 [ ] 的数值阵列对象；通过函数 randn () 和 rand ()，用户可以创建一个所有阵列元素为随机数的数值阵列对象，其中函数 randn () 产生的随机数服从正态分布。以上的这些函数，由用户输入参数的个数确定所创建阵列的维数，假如给定 n 个输入参数，则生成一个 n 维的阵列对象。下面是一个简单的例子：

```
B = randn (2, 3, 2);
cout << " B =" << B << endl;
```

执行这段代码，将显示如下结果：

```
B =[
    (:,:, 1) =
    [
          -0.4326     0.1253     -1.1465
          -1.6656     0.2877      1.1909
       ]
    (:,:, 2) =
       [
          1.1892    0.3273   -0.1867
         -0.0376    0.1746    0.7258
       ]
    ]
```

而函数 magic () 和 eye () 仅可用于生成二维阵列，其中 eye () 用于生成一个单位矩阵对象，magic () 用于生成一个特殊的方阵对象，其行元素和、列元素和、对角线元素和均相等，例如

```
B = magic (5);
cout << " B =" << B << endl;
```

执行这段代码，将显示如下结果：

```
B =
[
    17    24    1     8    15
    23     5    7    14    16
     4     6   13    20    22
    10    12   19    21     3
    11    18   25     2     9
```

```
]
```

- 调用数学函数来创建数值阵列对象。这一点与 MATLAB 非常地类似，在 MATLAB C++数学函数库中大部分的数学函数和运算符均将产生一个新的阵列对象，例如下面的代码：

```
mwArray A, B;
A = magic (5);
B = eye (5);
mwArray C = A * B;
```

其中C为一个新的数值阵列对象，它为阵列对象A和B的乘积。

- 通过拼接已有的数值阵列来创建新的数值阵列对象，这主要是通过函数

```
horzcat ();
vertcat ();
cat ();
```

来完成的，其中函数horzcat () 用于完成水平拼接操作，函数vertcat () 用于完成垂直拼接操作，函数cat () 用于完成拼接创建多维阵列。例如在MATLAB环境中定义一个如下形式的阵列

```
A = [ 1 2 3 4 5 ]
```

直接在MATLAB命令提示符下键入命令

```
A = [ 1 2 3 4 5 ]
```

即可，回车后MATLAB将显示

```
A =

    1 2 3 4 5
```

而在C++中，通过MATLAB C++数学函数库则必须通过函数horzcat () 来构造，如下：

```
mwArray A;
A = horzcat ( 1, 2, 3, 4, 5 );
cout << " A = " << A << endl;
```

执行这段代码，将显示如下结果：

```
A = [
     1 2 3 4 5
    ]
```

同理，对于构造列向量也相同，不过使用的函数为vertcat ()。通过联合使用函数horzcat () 和函数vertcat () 可以创建二维的数值阵列对象，例如：

```
mwArray A = vertcat ( horzcat (1, 2, 3), horzcat (4, 5, 6) );
mwArray B = vertcat ( horzcat (1, 2, 3), horzcat (4, 5, 6) );
mwArray C = vertcat ( A, B );
```

执行这段代码，将显示如下结果：

```
C = [
      1 2 3;
      4 5 6;
      1 2 3;
      4 5 6
    ]
```

函数 cat () 与函数 horzcat () 和函数 vertcat () 存在较大的不同，其主要功能是在指定的维数上，将若干个阵列拼接为一个单独的阵列，例如：

```
mwArray A = vertcat ( horzcat (1, 2, 3), horzcat (4, 5, 6) );
mwArray B = vertcat ( horzcat (1, 2, 3), horzcat (4, 5, 6) );
mwArray C = cat (3, A, B );
```

执行这段代码，将显示如下结果：

```
C = [
(:, :, 1) =
    [
        1 2 3;
        4 5 6
    ]
(:, :, 2) =
    [
        1 2 3;
        4 5 6
    ]
]
```

函数的第一个参数为指定的拼接的维数。如果指定的维数大于用户指定的阵列的维数，则函数自动地添加一个大小为 1 的维，例如

```
mwArray A = vertcat ( horzcat (1, 2, 3), horzcat (4, 5, 6) );
mwArray B = vertcat ( horzcat (1, 2, 3), horzcat (4, 5, 6) );
mwArray C = cat (4, A, B );
```

执行这段代码，将显示如下结果：

```
C = [
    (:, :, 1, 1) =
        [
            1 2 3;
            4 5 6
        ]
    (:, :, 1, 2) =
        [
            1 2 3;
            4 5 6
```

```
            ]
        ]
```

- 通过赋值操作完成数值阵列对象的创建。这种方法非常简单，例如

```
mwArray A = 14;
```

该语句创建了一个大小为 1×1 的数值阵列。

2. 稀疏矩阵的创建

稀疏矩阵是 MATLAB V5.X 版本所提供的一种新的阵列数据类型，它提供了一种高效的存储包含大量零元素的二维阵列的方式，在本书的第一章我们已经对它进行了详细的介绍。在 MATLAB C++数学函数库 2.0 版本中，提供了对稀疏矩阵的支持，这与 1.2 版本相比是一个明显的功能增强。表 8.3 是函数库中一些稀疏矩阵的创建函数和基本的稀疏矩阵操作函数的列表。

表 8.3　有关稀疏矩阵的库函数

函数原型	功　　能
sparse ()	创建一个稀疏矩阵
full ()	将一个稀疏矩阵转化为一个满矩阵
spones ()	将所有的非零的稀疏矩阵元素替换为 1
sprand () sprandn () sprandnsym ()	将所有的非零的稀疏矩阵元素用随机数替代
spconvert ()	将一个文本文件转换为稀疏矩阵
speye ()	创建一个稀疏的单位矩阵
spdiags ()	从一个数值阵列中提取数据，并使用这些数据创建一个稀疏矩阵
nnz ()	察看某个数值矩阵中非零元素的个数
any () 或 all ()	判断是否数值矩阵的元素皆为零元素或皆为非零元素
nzmax ()	确定稀疏矩阵中非零元素的个数
nonzeros ()	获取稀疏矩阵中所有的非零元素并存储为一个向量
spfun ()	对稀疏矩阵中的所有非零元素应用一个函数

总的来说，可以通过两种方法创建稀疏矩阵：

- 将一个现有的数值矩阵转换为稀疏矩阵，这主要是通过函数 sparse () 来完成的，例如

```
mwArray A, B;
A = eye (10);
cout << A << endl;
B = sparse (A);
cout << B << endl;
```

执行这段代码，将显示如下的内容：

```
[
```

```
    1  0  0  0  0  0  0  0  0  0
    0  1  0  0  0  0  0  0  0  0
    0  0  1  0  0  0  0  0  0  0
    0  0  0  1  0  0  0  0  0  0
    0  0  0  0  1  0  0  0  0  0
    0  0  0  0  0  1  0  0  0  0
    0  0  0  0  0  0  1  0  0  0
    0  0  0  0  0  0  0  1  0  0
    0  0  0  0  0  0  0  0  1  0
    0  0  0  0  0  0  0  0  0  1
]

    (1, 1) 1
    (2, 2) 1
    (3, 3) 1
    (4, 4) 1
    (5, 5) 1
    (6, 6) 1
    (7, 7) 1
    (8, 8) 1
    (9, 9) 1
    (10, 10) 1
```

- 从一定的数据中创建一个稀疏矩阵，这同样是通过函数 sparse () 来完成的，不过在使用函数 sparase 时，必须按以下的格式提供参数：

  sparse (i, j, k, m, n [, nzmax])

  其中 i,j 和 k 为三个具有同样长度的阵列对象，i 中包含了稀疏矩阵中非零元素的行的下标，j 中包含了稀疏矩阵中非零元素的列的下标，k 中则包含了稀疏矩阵中非零元素的值；m，n 和 nzmax 为三个数量，m 和 n 决定了所创建稀疏矩阵的行数和列数，nzmax 为一个可选参数，用于确定稀疏矩阵中可以存放的最多的非零元素的个数。例如

```
//数据的定义
double inums [] = {3, 4, 5, 4, 5, 6};
double jnums [] = {4, 3, 3, 5, 5, 4};
double knums [] = {1, 2, 3, 4, 5, 6};

// 创建阵列对象
mwArray S;
mwArray i (1, 6, inums, NULL);
mwArray j (1, 6, jnums, NULL);
mwArray k (1, 6, jnums, NULL);

// 创建稀疏矩阵
S = sparse (i, j, k, 8, 8, 10);
```

```
cout << S << endl;
cout << full (S) << endl;
```

执行这段代码，将显示如下内容：

```
(4, 3)    2
(5, 3)    3
(3, 4)    1
(6, 4)    6
(4, 5)    4
(5, 5)    5
 0  0  0  0  0  0  0
 0  0  0  0  0  0  0
 0  0  0  1  0  0  0
 0  0  2  0  4  0  0
 0  0  3  0  5  0  0
 0  0  0  6  0  0  0
 0  0  0  0  0  0  0
 0  0  0  0  0  0  0
```

3. 字符阵列的创建

一般我们对于一维的字符阵列称之为字符串，而对于多维的字符阵列则称之为字符串数组。在 MATLAB 中，字符串数组要求其中的每一个字符串拥有同样的长度，在 MATLAB C++数学函数库中，字符串数组有着同样的要求，不过这一点无需用户在编制程序时专门指出，函数库中的字符阵列创建函数在创建字符串数组时，会自动在长度不足的字符串后添加空格，使数组中所有的字符串拥有同样的长度。

在 MATLAB C++数学函数库中，提供了若干个用于创建字符阵列对象的库函数，分别如下：

```
mwArray (const char * )
char_func ();
str2mat ();
strcat ();
strvcat ();
num2str ();
int2str ();
```

总的来说，可以将这些函数分为三类，如下：

- 使用类 mwArray 的构造函数来创建字符阵列对象，即使用函数

  ```
  mwArray (const char * )
  ```

  来创建字符阵列对象，这也是最简单的一种字符阵列对象的构造方法。例如

  ```
  mwArray A (" MATLAB C++ Math Library");
  cout << " A = " << A << endl;
  ```

  执行这段代码，将显示如下内容：

  'MATLAB C++ Math Library'

- 通过将数值阵列转换为字符阵列来创建字符阵列对象，这主要是通过以下三个函数来实现的：

```
char_func ();
num2str ();
int2str ();
```

其中函数 char_func () 可以按 ASCII 码表的数值和字符的对应关系将一个数值阵列转换为字符阵列，例如

```
// 声明两个 mwArray 类对象
mwArray A, B;
// 将 A 初始化为一个数值阵列
A = horzcat (109, 121, 32, 115, 116, 114, 105, 110, 103);
// 使用函数 char_func () 将 A 转换为字符阵列
B = char_func (A);
// 输出字符阵列 B
cout << B << endl;
```

执行这段代码，将显示如下内容：

```
'my string'
```

此外通过函数 char_func ()，用户还可以轻松地构造多维的字符阵列，例如

```
mwArray A (" MATLAB API");
mwArray B (" MATLAB C++ Math Library ");
mwArray C = char_func (Z, Y);
cout << C << endl;
```

执行这段代码，将显示如下内容：

```
[
  'MATLAB API ' ;
  'MATLAB C++ Math Library ' ;
]
```

函数 num2str () 可以直接将一个数值阵列转换为一个字符阵列，例如

```
// 声明两个 mwArray 类对象
mwArray A, B;
// 将 A 初始化为一个数值阵列
A = horzcat (109, 121, 32, 115, 116, 114, 105, 110, 103);
// 使用函数 num2str () 将 A 转换为字符阵列
B = num2str (A);
// 输出字符阵列 B
cout << B << endl;
```

执行这段代码，将显示如下内容：

```
'109 121 32 115 116 114 105 110 103'
```

函数 int2str () 完成的工作与函数 num2str () 类似，只不过在转换前需要进行四舍五入的取整操作，例如

```
// 声明两个 mwArray 类对象
mwArray c, b;
// 将 c 初始化为一个数值阵列
c = horzcat (1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9);
// 使用函数 int2str () 将 A 转换为字符阵列
b = int2str (c);
cout << b << endl;
```

执行这段代码，将显示如下内容：

```
'1 1 1 1 1 2 2 2 2 2'
```

· 通过拼接已有的字符阵列来构造新的字符阵列对象，这主要是通过函数

```
str2mat ();
strcat ();
strvcat ();
```

来完成的，其中函数 str2mat () 和函数 strvcat () 的功能相同，它们都是将输入的字符阵列按行垂直拼接起来，并且通过添加空格使阵列中所有的字符串拥有同样的长度，它们之间惟一的不同是函数 str2mat () 不忽略输入字符阵列中的空阵列，而函数 strvcat () 将忽略空阵列，例如

```
// 声明两个 mwArray 类对象
mwArray a, b;
// 使用函数 str2mat () 构造字符阵列
a = str2mat (" abcdefg","","  lmnopq");
// 使用函数 strvcat () 构造字符阵列
b = strvcat (" abcdefg","","  lmnopq");
// 输出字符阵列
cout << " a = " << endl;
cout << a << endl;
cout << " b = " << endl;
cout << b << endl;
```

执行这段代码，将显示如下内容：

```
a =
 [
  'abcdefg' ;
  '        ' ;
  'lmnopq ' ;
 ]

b =
 [
  'abcdefg' ;
  'lmnopq ' ;
 ]
```

函数 strcat () 的功能是将输入的字符阵列在水平方向上拼接起来，例如

```
// 声明一个 mwArray 类对象
mwArray a;
// 使用函数 str2mat () 构造字符阵列
a = strcat (" abcdefg",""," lmnopq");
// 输出字符阵列
cout << " a = " << endl;
cout << a << endl;
```

执行这段代码，将显示如下内容：

```
a =
'abcdefglmnopq'
```

4. 单元阵列的创建

单元阵列是 MATLAB V5.X 版本所提供的一种新的阵列数据类型，它提供了一种将多个不同类型的阵列数据存放在一起的手段，在本书的第一章我们已经对它进行了详细的介绍。在 MATLAB C++数学函数库 2.0 版本中，提供了对单元阵列的支持，这与 1.2 版本相比是一个明显的功能增强，其中用于创建单元阵列对象的函数主要有以下一些：

```
cell ();
cellstr ();
cellhcat ();
struct2cell ();
num2cell ();
```

总的来说，用户可以通过四种方法来创建单元阵列对象，分别为：

- 使用单元阵列的创建函数来构造单元阵列对象，这主要是通过函数 cell () 来完成的，例如

```
// 声明 mwArray 类对象
mwArray d;
// 通过函数 cell () 构造一个 3×3×3 的单元阵列
d = cell (3, 3, 3);
cout << " d =" <<endl;
cout << d <<endl;
```

执行这段代码，将显示如下内容：

```
d =
(:,:, 1) =
        []      []      []
        []      []      []
        []      []      []
(:,:, 2) =
```

```
        []      []      []
        []      []      []
        []      []      []
(:,:, 3) =
        []      []      []
        []      []      []
        []      []      []
```

- 使用转换函数构造单元阵列对象，这主要是通过以下三个函数

```
cellstr ();
struct2cell ();
num2cell ();
```

来完成的，其中函数 cellstr () 可以将一个字符阵列转换为一个单元阵列，例如

```
mwArray b, d;
b = strvcat (" abcdefg","","lmnopq");
d = cellstr (b);
cout << " d=" <<endl;
cout << d <<endl;
```

执行这段代码，将显示如下内容：

```
d=
    'abcdefg'
    'lmnopq'
```

函数 num2cell () 可以将一个数值阵列转换为一个单元阵列，例如

```
mwArray c;
c = ones (2, 3);
c = num2cell (c);
cout << " c=" <<endl;
cout << c <<endl;
```

执行这段代码，将显示如下内容：

```
c=
[1]     [1]     [1]
[1]     [1]     [1]
```

函数 struct2cell () 可以将一个结构体转换为单元阵列。

- 使用拼接的方法来构造单元阵列对象，这主要是通过函数 cellhcat () 来完成的，该函数的使用方法与 MATLAB 中的 {} 运算符非常相似，例如在 MATLAB 中我们可以通过在命令提示符下键入如下命令

```
? e = {'join us' ones (5) eye (6) 8}
```

来创建一个单元阵列 e，而在 C++语言中，通过以下的代码

```
mwArray e;
e = cellhcat (" join us", ones (5), eye (6), 8);
```

```
cout << " e=" <<endl;
cout << e <<endl;
```

完成同样的工作，执行这段代码，将显示如下内容：

```
e=
  'join us' [5x5 double] [6x6 double] [8]
```

如果用户希望创建多行或多维的单元阵列，可以配合使用函数 vertcat () 和函数 cat () 即可。例如

```
mwArray e, f;
e = vertcat (cellhcat (" join us", ones (5)), cellhcat (eye (6), 8));
f = cat (3, cellhcat (" join us", ones (5)), cellhcat (eye (6), 8));
cout << " e=" <<endl;
cout << e <<endl;
cout << " f=" <<endl;
cout << f <<endl;
```

执行这段代码，将显示如下内容：

```
e=
    'join us'        [5x5 double]
     [6x6 double]  [8]
f=
(:,:, 1) =
    'join us'          [5x5 double]
(:,:, 2) =
     [6x6 double]  [8]
```

・通过赋值来完成单元阵列对象的构造，例如

```
mwArray g;
g (2, 2) = cellhcat (" join us");
cout << " g=" <<endl;
cout << g <<endl;
```

执行这段代码，将显示如下内容：

```
g=
     []        []
     []         'join us'
```

5. 结构体阵列的创建

结构体阵列是 MATLAB V5. X 版本所提供的一种新的阵列数据类型，在本书的第一章我们已经对这种阵列类型进行了详细的介绍。在 MATLAB C++数学函数库 2.0 版本中，提供了对结构体阵列的支持，这与 1.2 版本相比是一个明显的功能增强。总的来说，在 MATLAB C++数学函数库中，共提供了三种构造结构体阵列对象的方法，分别如下：

- 使用结构体阵列的构造函数来创建结构体阵列对象，这主要是通过函数 struct_func () 来完成的，例如

```
mwArray a, b;
a = struct_func (" first_name",        // field name
                 " Wang",               // value
                 " last_name",          // field name
                 " xiao ming",          // value
                 " age",                // field name
                 17);                   // value
cout << " a = " << endl;
cout << a << endl;
```

执行这段代码，将显示如下内容：

```
a =
    first_name: 'Wang'
     last_name: 'xiao ming'
           age: 17
```

- 通过转换函数将单元阵列对象转换为结构体阵列对象，这主要是通过函数 cell2struct () 来完成的，例如

```
mwArray a, b, c;
// 构造单元阵列对象，用于存放值
a = cellhcat (" Wang xiaoming",
             17,
             " mingming");
cout << a << endl;
//构造单元阵列对象，用于存放域名
b = cellhcat (" name",
              " age",
              " nickname");
// 将单元阵列对象转换为结构体阵列对象
c = cell2struct (a, b, 2);
cout << " c=" <<endl;
cout << c << endl;
```

执行这段代码，将显示如下内容：

```
     'Wang xiaoming' [17] 'mingming'
c=
         name: 'Wang xiaoming'
          age: 17
     nickname: 'mingming'
```

- 通过赋值操作来创建结构体阵列对象，例如

```
mwArray c;
c (3, 3) = struct_func (" name"," Wang xiaoming"," age"," 17");
cout << " c=" <<endl;
cout << c << endl;
```

执行这段代码，将显示如下内容：

```
c=
3x3 struct array with fields:
    name
    age
```

8.2.2 阵列对象的索引操作

在上一小节中，我们对 MATLAB C++数学函数库中各种类型的阵列对象的创建操作进行了说明，在本小节中，我们将讲述 MATLAB C++数学函数库中另一种关于阵列对象的基本操作，即索引操作，通过索引操作用户可以方便地完成对阵列对象中元素的访问、修改和删除任务。在 MATLAB 中，对于三种不同类型的阵列，即数值阵列、单元阵列和结构体阵列提供了三种不同的索引操作、即小括号 ()、大括号 {} 和域名 (field name)。相应的，在 MATLAB C++数学函数库中，同样对三种不同的阵列对象提供了不同的索引操作，它们之间的对应关系如下：

数值阵列对象 —— 运算符 ()

单元阵列对象 —— mwArray:: cell ()

结构体阵列对象 —— mwArray:: field ()

其中运算符 () 和函数 mwArray:: cell () 的输入参数为数值，而函数 mwArray:: field () 的输入参数为阵列的域名。下面我们将对它们进行分别讲述。

在进行详细讲述之前，用户必须清楚两个基本概念，即

- 索引和下标。参见下面的表达式：

  A (2, 4)

  该式为一个典型的索引表达式，其中 A 为一个 mwArray 类对象，数值 2 和 4 统称为索引，而 (2, 4) 作为一个整体称为下标，并且是一个二维下标，通常小 () 中用逗号分隔开的独立的索引的个数称为下标的维数。
- 阵列对象元素的存储。关于这一点我们在前面已经多次提到，阵列对象元素的存储方法与 C++语言中的数组的存储方法截然不同，在 C++语言中，数组元素为按行存储，而阵列对象元素则恰好相反，为按列存储。例如一个二维的大小为 $[d_1, d_2]$ 的阵列对象，其中第 (i, j) 个元素的偏移量为 $(j-1)*d_1+i$ (注：阵列对象的元素的索引值的起始值为 1，而 C++数组则为 0)；对于多维的阵列与此类似，假设一个 n 维阵列对象的大小为 $[d_1, d_2, d_3, \cdots, d_n]$，则其中第 $(a_1, a_2, a_3, \cdots, a_n)$ 个元素的偏移量可以由下面的公式进行计算：

  $(a_n-1)(d_n-1)(d_{n-2})\ldots(d_1)+(a_{n-1})(d_{n-2})\ldots(d_1)+\ldots+(a_2-1)(d_1)+a_1$

1. 数值阵列对象的索引操作

根据下标的维数和类型，可以将数值阵列对象索引操作的运算符（）分为三种类型，即使用一维下标的索引操作，使用多维下标的索引操作和使用逻辑下标的索引操作，下面我们将分别进行说明。

· 使用一维下标的索引操作

通过使用不同形式的一维下标表示方式，用户可以获得不同的索引结果。当一维下标中仅包含单一的索引元素时，所获取的索引结果同样为单一的阵列元素，例如令

```
A =
    1  2  3
    4  5  6
    7  8  9
```

则索引表达式 A（2）的结果为 4；当一维下标由一个向量组成时，则所获取的索引结果同样为一个向量，例如索引表达式 A (horzcat (3, 5, 7)) 和索引表达式 A (vertcat（3, 5, 7)）的结果分别为

```
7 5 3


7
5
3
```

当一维下标由一个矩阵表示时，则索引表达式返回的结果为一个同样大小的矩阵，例如矩阵 B 为

```
B =
    3  5
    7  9
```

则索引表达式 A（B）的索引结果为

```
7  5
3  9
```

此外，用户还可以通过函数 colon（）作为索引来获取所有的阵列元素，例如索引表达式 A（colon（)）的索引结果为

```
1
4
7
2
5
8
3
6
9
```

· 使用多维下标的索引操作

在使用多维下标时，下标中的第一个索引代表阵列的行索引值，第二个索引代表阵列的列索引值，第三个索引代表阵列的页面索引值，第四个第五个索引以此类推，例如索引表达式 A（2，3）的索引结果为 5，这里阵列 A 的值同上。当用户希望从阵列中提取出若干个元素并表示为向量时，可以用多维下标表示为如下形式：

A（horzcat（2，3），1）或 A（1，horzcat（2，3））

它们的索引结果分别为

```
4
7


2  3
```

将函数 colon（）作为行或列的索引，可以获得阵列中整行或整列的元素，例如索引表达式

A（colon（），2）或 A（2，colon（））

它们的索引结果分别为

```
2
5
8


4 5 6
```

如果将所有的索引用向量的形式表示，则所获取的索引结果为阵列形式，例如索引表达式

A（horzcat（1，2），horzcat（1，2））

的结果为

```
1 2
4 5
```

对于索引维数大于 2 的索引表达式，使用方法与二维情况类似，令

```
B =
Page 1
    1  2  3
    4  5  6
    7  8  9
Page 2
    10  11  12
    13  14  15
    16  17  18
```

则索引表达式 B（colon（），colon（），2）和 B（1，colon（），colon（））的索引结果分别为

```
10  11  12
13  14  15
16  17  18
Page 1
    1  2  3
```

```
Page 2
    10  11  12
```

- 使用逻辑下标的索引操作

所谓逻辑下标就是指所有的索引的元素均由逻辑值零和一构成的下标，例如矩阵

```
B =
        0  1  0
        1  0  1
        0  1  0
```

就是一个逻辑下标，令

```
A =
        1  2  3
        4  5  6
        7  8  9
```

则索引表达式A（logical（B））的结果为

```
    2
    4
    6
    8
```

当逻辑索引表达式为如下形式

A（logical（horzcat（1，0，1）），logical（horzcat（0，1，0）））

时，索引结果为

```
        2
        8
```

这里logical（horzcat（1，0，1））表示行索引，含义为选取阵列A的第一和第三行，而logical（horzcat（0，1，0））表示列索引，含义为选取阵列A的第二列。在使用逻辑索引时，同样可以使用函数colon()对阵行或阵列的阵列元素进行访问，例如索引表达式

A（logical（colon（），logical（horzcat（0，1，1）））

的索引结果为

```
    2  3
    5  6
    8  9
```

2. 单元阵列对象的索引操作

单元阵列是MATLAB V5.X版本所提供的一种新型的阵列对象，其构成与常规的阵列对象（与稀疏矩阵相对）相同，可以为任意维数、任意大小，只不过阵列中的每一个元素均为一个阵列对象，称为单元，各个单元所包含的阵列对象之间的类型、维数和大小均可以各不相同，并且允许嵌套，即单元阵列的某个单元所包含的阵列对象仍然是一个单元阵列，有关单元阵列在本书的第一章有详细的描述。

在MATLAB中，对单元阵列的索引操作提供了两种索引运算符，即常规的索引运算符 () 和专用索引运算符 {}。相应地，在MATLAB C++数学函数库中同样提供了两种关于单元阵列对象的索引操作，分别为运算符 () 和类mwArray的成员函数cell ()。

单元阵列对象的索引操作同数值阵列对象的索引操作方法基本类似，既可以通过用向量形式表示的下标来获取向量形式的索引结果，也可以通过矩阵形式表示的下标来获取矩阵形式的索引结果，同时还可以通过函数colon () 来获取整行或整列的阵列内容，不过这里必须注意的一点是使用标准索引符 () 和使用类mwArray的成员函数cell () 进行索引所返回的结果并不相同，使用标准索引符 () 时，返回的结果仍然为单元阵列，而使用类mwArray的成员函数cell () 返回的结果却是与单元阵列对象中某个指定单元所包含的阵列对象类型相同的阵列对象，下面我们将举例说明单元阵列对象的各种索引操作。

令

```
A.cell (1, 1) = vertcat (horzcat (1, 3), horzcat (2, 4));
A.cell (1, 2) = complex (-2, 5);
A.cell (2, 1) = " MATLAB C++ Math Library";
A.cell (2, 2) = 3;
```

- 对单元阵列单个单元的索引操作，这主要是通过索引运算符 () 来完成的，例如索引表达式

  ```
  B = A (2, 1)
  ```

  其中B为一个mwArray类对象，执行该索引表达式返回的结果为

  ```
  B=
      'MATLAB C++ Math Library'
  ```

  并且阵列对象B为一个1×1的单元阵列。

- 访问单元阵列对象的一个子集，这主要是通过索引运算符 () 来完成的，例如索引表达式

  B = A (2, colon ()) 和 C = A (1, horzcat (1, 2))

  其中B和C均为mwArray类对象，执行这两个索引表达式返回的结果分别为

  ```
  B=
     [1x23 char]      [3]
  C=
     [2x2 double]     [-2.0000+ 5.0000i]
  ```

  并且阵列对象B和C均为1×2的单元阵列。

- 访问单元阵列中某个单元的内容，这主要是通过类mwArray的成员函数cell () 来完成的，例如索引表达式

  ```
  B = A.cell (2, 1)
  ```

  其中B为一个mwArray类对象，执行该索引表达式返回的结果为

  ```
  B=
  'MATLAB C++ Math Library'
  ```

  并且阵列对象B为一个字符串阵列对象。

- 获取单元阵列中某个单元的内容的子集，这主要是通过类 mwArray 的成员函数 cell () 和索引运算符 () 联合作用来完成的，例如索引表达式

```
B = A.cell (2, 1) (1, horzcat (5, 9, 15, 20));
```

其中 B 为一个 mwArray 类对象，执行该索引表达式返回的结果为

```
B=
'A+hr'
```

并且阵列对象 B 为一个字符串阵列对象。

- 对于包含嵌套的单元阵列中嵌套单元的访问，这主要是通过重复使用类 mwArray 的成员函数 cell () 来完成的。令

```
mwArray B, A;
A.cell (1, 1) = vertcat (horzcat (1, 3), horzcat (2, 4));
A.cell (1, 2) .cell (1, 1) = complex (-2, 5);
A.cell (1, 2) .cell (1, 2) = " abcdefg";
A.cell (1, 2) .cell (2, 1) = 4;
A.cell (1, 2) .cell (2, 2) .cell.(1, 1) = 10;
A.cell (1, 2) .cell (2, 2) .cell (1, 2) = " rstuvw";
A.cell (2, 1) = " MATLAB C++ Math Library";
A.cell (2, 2) = 3;
```

则执行以下代码

```
B = A.cell (1, 2) .cell (2, 2) .cell (1, 2);
cout << " B=" <<endl;
cout << B << endl;
```

返回的结果为

```
B=
'rstuvw'
```

并且阵列对象 B 为一个字符串阵列对象。这里通过多次使用成员函数 cell () 访问到了单元的最底层。如果索引表达式为如下形式：

```
B = A.cell (1, 2) .cell (2, 2);
```

则返回的结果为

```
B=
[10] 'rstuvw'
```

阵列对象 B 为一个单元阵列对象。

3. 结构体阵列对象的索引操作

MATLAB 结构体 (structure) 是 MATLAB V5.X 中的一种新型的阵列类型，其定义与 C 语言和 C++语言中结构体的定义类似，是由一系列不同的域 (field) 组成，每一个域均可以为不同类型的 MATLAB 阵列，甚至可以为另外一个 MATLAB 结构体。与常规的 MATLAB 阵列相同，MATLAB 结构体同样为面向阵列的数据类型，单一的结构体

对象实际为一个 1×1 的结构体阵列对象，这与 MATLAB 中将数值 5 视为一个 1×1 数值阵列对象的概念相同。同理，用户可以构造多维的结构体阵列对象，其每一个元素均为一个结构体对象，不过这里必须注意的一点是在结构体阵列对象中，每一个结构体对象所包含的域必须完全相同，当对阵列中某一个结构体对象进行域的增加和删除操作时，结构体阵列中所有的结构体对象将同时增加或删除某一个域。

在 MATLAB C++数学函数库中，为结构体阵列对象的索引操作提供了两种不同的手段，分别用于完成不同的任务，在使用标准的索引操作运算符（）时，完成的主要功能是对结构体阵列中结构体对象的索引操作，而在使用类 mwArray 的成员函数 field () 时，可以通过将域名作为参数完成对结构体阵列中结构体对象的域的索引操作。例如存在一个名为 peoples 的结构体阵列，大小为 3×3×3，其中 peoples（2，2，2）所包含的内容如下：

```
peoples (2, 2, 2) .field (" firstname") = " Wang";
peoples (2, 2, 2) .field (" lastname") = " xiaoming";
peoples (2, 2, 2) .field (" age") = 17;
peoples (2, 2, 2) .field (" birthday") .field (" year") = 1975;
peoples (2, 2, 2) .field (" birthday") .field (" month") = 11;
peoples (2, 2, 2) .field (" birthday") .field (" day") = 15;
```

该结构体对象包含四个域，分别为 firstname，lastname，age 和 birthday，其中域 birthday 又为一个结构体对象，包含三个域，分别为 year，month 和 day。通过索引表达式

```
B = peoples (2, 2, 2);
```

用户可以获得指定的结构体对象，其中 B 被声明为一个 mwArray 类对象，执行该表达式将返回如下结果：

```
B=
    firstname: 'Wang'
      lastname: 'xiaoming'
           age: 17
      birthday: [1x1 struct]
```

这里 B 为一个 1×1 的结构体阵列对象。通过索引表达式

```
B = peoples (2, 2, 2) .field (" birthday");
```

用户可以获得指定结构体阵列中指定结构体对象中指定域的内容，执行该索引表达式将返回如下结果：

```
B=
    year: 1975
    month: 11
    day:   15
```

这里 B 为一个结构体对象，因为 birthday 域的内容为一个结构体对象，如果索引表达式为

```
B = peoples (2, 2, 2) .field (" firstname");
```

执行该表达式将返回下结果：

```
B=
'Wang'
```

这里 B 为一个字符类型的阵列对象，因为 firstname 域的内容为一个字符阵列。

8.3 应用程序的编写

在本节中，我们将对 MATLAB C++数学函数库中数学运算符和库函数的使用分别进行简要说明，然后给出一个完整的基于 MATLAB C++数学函数库编写的范例程序，最后简要讲述如何在 Microsoft Visual C++ 6.0 集成环境中建立基于 MATLAB C++数学函数库的应用程序。

8.3.1 数学运算符的使用

在 MATLAB 中，为了方便用户对阵列的数学运算，系统提供了大量的数学运算符供用户使用，分别如下：

+，－，.*，./，.\，.^，*，/，\，^，'，.'

从总体上可以将这些数学运算符分为两类，即阵列运算符和矩阵运算符，一般来说，矩阵运算符的前方没有符号“.”，而阵列运算符的前方拥有符号“.”，例如运算符 * 和运算符 .* 分别是矩阵的乘法运算符和阵列的乘法运算符。矩阵运算符和阵列运算符在功能上存在着相当大的差异，矩阵运算符的定义为线性代数中矩阵数学运算的定义，仅仅适用于二维的阵列，即矩阵，所以称这些运算符为矩阵运算符，而阵列运算符则不同，它作用于阵列中的每一个元素，使用于任何维数的阵列，包括二维，不过要求运算符的左右两个操作数必须同维数，并且每一维的大小也相同。例如存在矩阵 A 和 B，它们的定义分别如下：

```
A = [1 1; 1 1];
B = [2 2; 2 2];
```

使用矩阵的乘法运算符，在 MATLAB 命令提示符下键入命令

```
C = A*B
```

回车后可以得到如下结果：

```
ans =
      4    4
      4    4
```

使用阵列的乘法运算符，在 MATLAB 命令提示符下键入命令

```
C = A.*B
```

回车后可以得到如下结果：

```
ans =
      2    2
      2    2
```

可见二者的计算结果完全不同。这里必须注意的是对于加法和减法没有矩阵运算符和阵

列运算符的区别，因为它们完成的功能相同，所以在MATLAB 中只提供了加法运算符+和减法运算符－，而没用提供 . +和 . －运算符。

在MATLAB C++数学函数库中，对所有的MATLAB 的数学运算符提供了对应函数。由于一些符号在C++语言中为非法的标示符，因此在MATLAB C++数学函数库中无法定义，只能使用函数加以替代，完成功能。表 8.4 为MATLAB 中数学运算符和MATLAB C++数学函数库中运算符及函数的对应关系表。

表 8.4　数学运算符对应关系

MATLAB 数学运算符	C++运算符	C++函数
+	+	plus ()
－	－	minus ()
*	*	mtimes ()
/	/	mrdivide ()
\	无	mldivide ()
^	^	mpower ()
.*	无	times ()
./	无	rdivide ()
.\	无	ldivide ()
.^	无	power ()
'	无	ctranspose ()
.'	无	transpose ()

它们的使用非常简单，与C++语言中普通运算符和函数的调用完全一致，不同的只是参数类型应为 mwArray 类对象，例如

```
static double data1 [] = { 1, 1, 1, 1 };
static double data2 [] = { 2, 2, 2, 2 };
mwArray A (2, 2, data1);
mwArray B (2, 2, data2);

// 矩阵乘法
mwArray C = mtimes (A, B);
cout << " C =" <<endl;
cout << C << endl;

// 阵列乘法
mwArray D = A + B;
cout << " D =" <<endl;
cout << D << endl;
```

执行该段代码可以得到如下结果：

```
C =
```

```
    [
        4    4;
        4    4
    ]
D =
     [
        3    3;
        3    3
    ]
```

除了MATLAB C++数学函数库提供的运算符外，用户还可以通过C++提供的重载机制对一些运算符进行重载，用以完成特定任务，例如用户希望对C++支持的自乘运算符 *= 进行重载，用以完成矩阵的自乘操作，可以用以下代码实现：

```
mwArray operator *= (mwArray &A, const mwArray &B)
{
    A = A * B;
    return A;
}
```

8.3.2 库函数的调用

在MATLAB C++数学函数库中总共提供了大约400个数学函数，其中每一个函数都对应于MATLAB中的一个数学函数，用于完成同样的功能，但是由于MATLAB语言语法和C++语言语法的区别，二者之间的调用形式不尽相同。在本小节中，我们将在分析MATLAB语言语法和C++语言语法区别的基础上，对MATLAB C++数学函数库中函数的调用进行讲述。

对于MATLAB中的这样一些函数，它们拥有固定个数的输入参数和单一的输出参数，对于它们，几乎无需任何的转换，就可以直接使用在C++语言中。这主要是因为在MATLAB和在MATLAB C++数学函数库中，对于这些函数调用的语法完全相同，例如函数log在MATLAB中的调用语法为

```
y = log (x)
```

而在MATLAB C++数学函数库中，可以使用同样的方法进行调用

```
y = log (x);
```

只不过在调用前必须将变量y和x声明为mwArray类对象。

对于MATLAB中拥有可选输入参数的函数来说，MATLAB C++数学函数库为函数的任何一种调用格式提供了一个相应的C++语言函数，用以完成同样的功能，并且函数名完全相同，惟一的区别是函数的输入参数的个数不同。这里充分地利用了C++语言的重载机制，根据参数个数的不同，选择调用不同的函数。对于这些函数，它们的调用语法在MATLAB和MATLAB C++数学函数库中几乎没有不同。例如函数linspace在MATLAB中的声明为

```
function y = linspace (d1, d2, n)
```

该函数拥有三个输入参数，其中参数n为一个可选参数，函数在MATLAB中有两种调用

方法，分别为：

```
y = linspace (a, b)
y = linspace (a, b, n)
```

而在MATLAB C++数学函数库中，对两种调用格式分别声明了一个函数，如下

```
mwArray linspace (const mwArray &a, const mwArray &b);
mwArray linspace (const mwArray &a, const mwArray &b, const mwArray &n);
```

它们的调用格式为

```
y = linspace (a, b);
y = linspace (a, b, n);
```

对于MATLAB中拥有多个可选输出的函数来说，在MATLAB C++数学函数库中的调用有较大的不同，这主要是因为在C++语言中，不支持函数拥有多个返回值，一般函数至多只能拥有一个返回值，如果用户一定要求函数能够返回多个值，惟一的办法是使用指针，通过指向相同的内存地址来完成数据的交换，从而实现多返回值。例如MATLAB函数find ()，其在MATLAB中的调用格式分别如下：

```
k = find (X);
[i, j] = find (X);
[i, j, v] = find (X);
```

在MATLAB C++数学函数库中，对三种调用格式分别声明了一个函数，如下

```
mwArray find (const mwArray &X);
mwArray find (mwArray *j, const mwArray &X);
mwArray find (mwArray *j, mwArray *v, const mwArray &X);
```

它们的调用格式为

```
k = find (X);
i = find (&j, X);
i = find (&j, &v, X);
```

其中i为第一个输出参数，j为第二个输出参数，v为第三个输出参数，这里通过取地址运算符&实现了内存的共享。这里必须说明的一点是在调用函数find前无需对变量i、j和k进行内存分配，不过必须将它们说明为mwArray类对象。

在MATLAB中存在这样的一些函数，它们可以拥有任意多个的输入参数，在MATLAB的函数定义中，它们的输入参数列表的最后一个部分为省略号…，例如MATLAB中函数horzcat的定义为如下形式：

```
B = horzcat (A1, A2, A3, …)
```

通过该函数，用户可以将任意多个的阵列按水平方式连接起来。在调用时，用户可以在()内输入任意多个的参数，例如

```
B = horzcat (A1, A2, A3, A4, A5, A6, A7, A8)
```

然而在C++语言中，并没有提供对可变长度输入参数列表的支持，同时这类函数也不能通过C++语言的重载机制来实现，因为在函数调用前，并不知道将有多少个输入参

数。在MATLAB C++数学函数库中，主要是通过类mwArray 的友员类mwVarargin 和提供默认的输入参数来完成这类函数在C++中的实现，例如函数horzcat 在MATLAB C++数学函数库中的定义为

```
mwArray cellhcat (const mwVarargin &in1,
                  const mwArray &in2=mwArray:: DIN,
                  const mwArray &in3=mwArray:: DIN,
                  ……
                  const mwArray &in32=mwArray:: DIN);
```

该定义中包含了32个输入参数，其中第一个参数为mwVarargin类型的变量，第二至第32个参数为mwArray类型的变量，并且在函数的定义中，为所有的32个输入变量提供了默认的参数取值，为mwArray::DIN，当用户调用该函数而没有提供这些参数或仅提供了这些参数的一部分时，函数将自动使用该值来作为默认的参数取值。DIN为类mwArray的一个公有成员数据，为mwArray类对象，其声明可以参见头文件dblmtrx.h。当用户在C++应用程序中调用该函数时，根据输入参数的多少可以分为两种情况：

第一种情况，当输入参数的个数不大于32个时，用户无需使用mwVarargin类型的变量，而可以简单地输入一系列的mwArray类型的变量作为输入参数，例如

```
B = horzcat (A1, A2, A3, A4, A5, A6, A7, A8)
```

其中变量B，A1，A2，A3，A4，A5，A6，A7和A8均为mwArray类型的变量。这里必须注意的一点是输入变量A1仍然对应的是mwVarargin的输入变量，只不过在mwVarargin类对象中只包含一个mwArray类对象时，mwArray类对象和mwVarargin类对象等价；

第二种情况，当输入参数的个数大于32时，则用户必须构造一个mwVarargin类型的对象，并将其作为第一个参数传递给函数。类mwVarargin的构造函数同样为一个输入参数个数可变的函数，其定义大致与函数horzcat的定义相同，包含一个mwVarargin类型和31个mwArray类型的输入变量，从而保证了mwVarargin类对象可以嵌套使用。当用户希望向函数输入68个输入参数时，则必须书写为如下形式：

```
horzcat (mwVarargin (mwVarargin (A1, A2,..., A32), A33, ..., A63),
    A64, A65, A66, A67, A68);
```

与拥有任意个输入参数的函数类似，在MATLAB中还存在一种可以拥有任意个输出参数的函数，例如函数size，其输入参数有几维，其输出参数就可以有几个：

```
[d1, d2,..., dn] = size (X)
```

其中n为阵列X的维数。这类函数在MATLAB C++数学函数库中的实现与拥有任意个输入参数的函数的实现方法基本类似，只不过使用友员类为mwVarargout，例如函数size在MATLAB C++数学函数库中的定义为如下形式：

```
mwArray size (mwVarargout varargout,
              const mwArray &X,
              const mwArray &dim=mwArray:: DIN);
```

其中X为维数为n的mwArray类对象，varargout为一个mwVarargout类对象，dim为一个拥有默认取值的可选的mwArray类型的输入参数。在MATLAB中，如下的函数调用语句

```
[d1, d2, d3, d4, …, dn] = size (X)
```

在C++应用程序中可以使用下面的语句调用

```
size (mwVarargout (d1, d2, d3, d4, …, dn), X);
```

其中X为维数为n的mwArray类对象。与拥有可选个输出参数调用方法不同的是这里传递给函数的第一个参数为一个mwVarargout类对象，而不是指向mwVarargout类对象的指针，这一点必须非常注意。

8.3.3　范例程序

程序example.cpp是一个非常简单的使用MATLAB C++数学函数库的应用程序，程序中分别调用了库函数lu ()、qr () 和svd () 对矩阵A、B和X进行了三角分解、QR分解和奇异值分解，程序的源代码如下：

```
#include " matlab.hpp"
#include <stdlib.h>      /* used for EXIT _ SUCCESS */

#ifdef GCC
  #ifndef EXIT _ SUCCESS
    #define EXIT _ SUCCESS 0
  #endif
#endif

static double data1 [] = { 2, 3, 2, 4, 4, 3, 4, 2, 4, 12, -1, 1, 2, 6, 2, 1 };
static double data2 [] = { 1, 2, 1, -1, 1, 1, -1, 2, -1, 0, 0, 1 };
static double data3 [] = { 1, 2, 1, 1, 1, -1, -1, 0, 0, -1, 2, 1 };

int main (void)
{
    // 对矩阵A进行三角分解
    mwArray A (4, 4, data1);
    cout << " A = " << endl;
    cout << A << endl;
    mwArray L, U;
    cout << " [L, U] = lu (A)" << endl;
    L = lu (&U, A);
    cout << " L = " << endl;
    cout << L << endl;
    cout << " U = " << endl;
    cout << U << endl;

    // 对矩阵B进行QR分解
    mwArray B (4, 3, data2);
```

```
    cout << " B = " << endl;
    cout << B << endl;
    mwArray Q, R;
    cout << " [Q, R] = qr (B)" << endl;
    Q = qr (&R, B);
    cout << " Q = " << endl;
    cout << Q << endl;
    cout << " R = " << endl;
    cout << R << endl;

    // 对矩阵 X 进行奇异值分解
    mwArray X (3, 4, data3);
    cout << " X = " << endl;
    cout << X << endl;
    mwArray S, V;
    cout << " [U, S, V] = svd (X)" << endl;
    U = svd (&S, &V, X);
    cout << " U = " << endl;
    cout << U << endl;
    cout << " S = " << endl;
    cout << S << endl;
    cout << " V = " << endl;
    cout << V << endl;

    return (EXIT_SUCCESS);
}
```

在 MATLAB 中使用命令

```
mbuild example.cpp
```

对程序进行编译后运行可以得到如下结果：

```
A =
  [
      2  4   4   2 ;
      3  3  12   6 ;
      2  4  -1   2 ;
      4  2   1   1
  ]
[L, U] = lu (A)

L =
  [
    0.50000  1.00000  0.41667  1.00000 ;
    0.75000  0.50000  1.00000  0.00000 ;
    0.50000  1.00000  0.00000  0.00000 ;
    1.00000  0.00000  0.00000  0.00000
```

```
]

U =
  1.0e+001 *
  [
    0.40000  0.20000   0.10000    0.10000 ;
    0.00000  0.30000  -0.15000    0.15000 ;
    0.00000  0.00000   1.20000    0.45000 ;
    0.00000  0.00000   0.00000   -0.18750
  ]

B =
 [
    1   1  -1 ;
    2   1   0 ;
    1  -1   0 ;
   -1   2   1
 ]

[Q, R] = qr (B)

Q =
  [
   -0.37796  -0.37796   0.75593   0.37796 ;
   -0.75593  -0.37796  -0.37796  -0.37796 ;
   -0.37796   0.37796  -0.37796   0.75593 ;
    0.37796  -0.75593  -0.37796   0.37796
  ]
R =
  [
   -2.64575  -0.00000   0.75593 ;
    0.00000  -2.64575  -0.37796 ;
    0.00000   0.00000  -1.13389 ;
    0.00000   0.00000   0.00000
  ]

X =
  [
   1   1  -1  -1 ;
   2   1   0   2 ;
   1  -1   0   1
]

[U, S, V] = svd (X)
U =
  [
```

```
    0.08850  -0.90288  -0.42069 ;
    0.92535  -0.08178   0.37018 ;
    0.36863   0.42204  -0.82825
  ]

S =
  [
    3.20792  0.00000  0.00000  0.00000 ;
    0.00000  2.13495  0.00000  0.00000 ;
    0.00000  0.00000  1.07297  0.00000
  ]

V =
  [
     0.71942  -0.30183  -0.47399  -0.40825 ;
     0.20114  -0.65889   0.72485  -0.00000 ;
    -0.02759   0.42290   0.39208  -0.81650 ;
     0.66424   0.54398   0.31016   0.40825
  ]
```

8.3.4 集成环境中 MATLAB C++数学函数库应用程序的建立

虽然 MATLAB 为用户提供了一个方便的用于配置系统和编译基于 MATLAB C++数学函数库应用程序的命令 mbuild，但是如果读者使用该命令编译过应用程序，仍然会感觉到相当的不方便。因为一个应用程序的建立过程是一个不断修改不断编译的过程，这样必然会导致在程序的建立过程中，用户不断在 MATLAB 环境和 C++语言环境间切换，非常麻烦，而且命令 mbuild 没有提供相应的应用程序调试功能，如果程序编译通过，但是在执行时发生错误，这种情况下只有通过调试才能很快地发现错误并加以改进。如果需要对应用程序加以调试，只有重新建立一个工程来完成程序的调试任务，极为不方便。下面我们将以 Microsoft Visual C++ 6.0 集成环境为例，给出一种在集成环境中完成程序的全部建立和调试任务的方法，其具体步骤如下：

第一步，进入 Microsoft Visual C++ 6.0 集成环境，选择 File 菜单中的 New 菜单项，将弹出一个如图 8.1 所示的对话框，提示用户希望创建的应用程序的类型，总的来说，可以选择三种类型的应用程序，即 MFC AppWizard (exe)、Win32 Application 和 Win32 Console Application，为了说明方便这里选择 Win32 Console Application，即 Win32 控制台程序，按要求输入文件名，并点击 OK 按钮；

第二步，完成以上操作后，VC++ 6.0 编译器将弹出如图 8.2 所示的对话框，提示用户选择创建 Win32 console Application 程序的类型，这里我们选择其中的最为简单的类型 An Empty project，然后选择 Finish 按钮，创建该项目。

第三步，在项目工程创建完毕之后，选择下拉式菜单 Tools 中的菜单项 Options，将弹出 Options 对话框，选择其中的 Directories 属性页，如图 8.3 在其中的 Show directories for 下拉式选项框中分别选择 Include Files 和 Library Files，在下部的编辑框中输入以下路径：

Files　Projects　Workspaces　Other Documents

ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
Extended Stored Proc Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
New Database Wizard
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library

Project name:
Location:
F:\程序调试\
Create new workspace
Platforms:
Win32
Cancel

图 8.1　VC6.0 File New 对话框

MATLAB 根目录\EXTERN\INCLUDE

MATLAB 根目录\EXTERN\LIB

MATLAB 根目录\EXTERN INCLUDE\CPP

然后选择 OK 按钮：

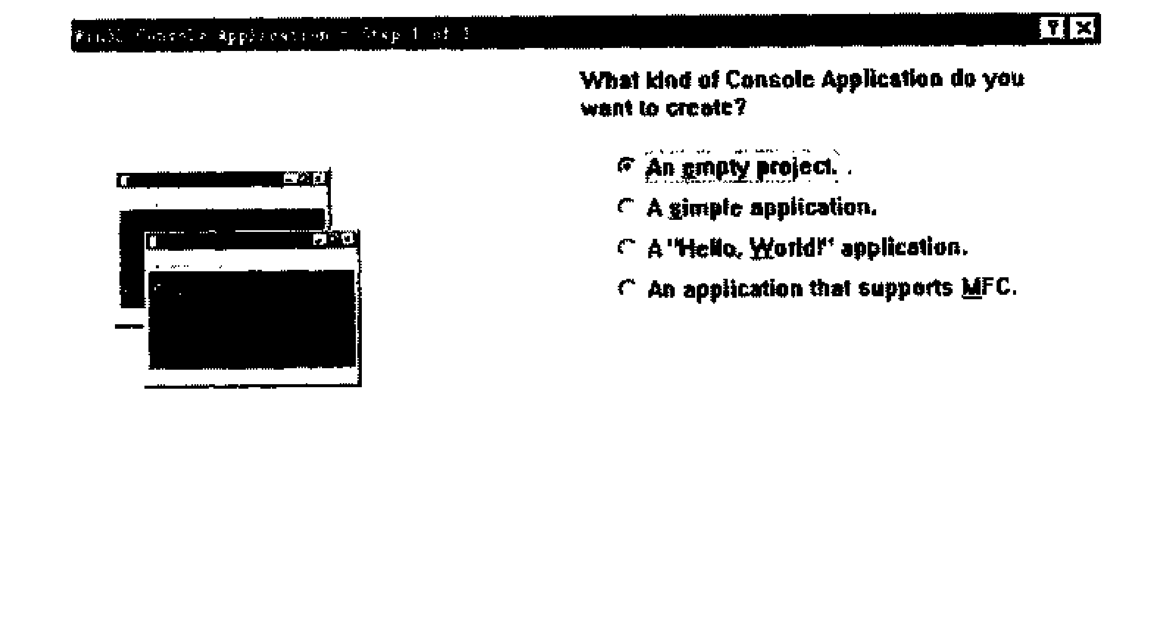


图 8.2　Win32 console Application 程序的类型选择对话框

第四步，在 DOS 命令框状态下，进入用户安装 Microsoft VC++ 6.0 的目录，如 d：

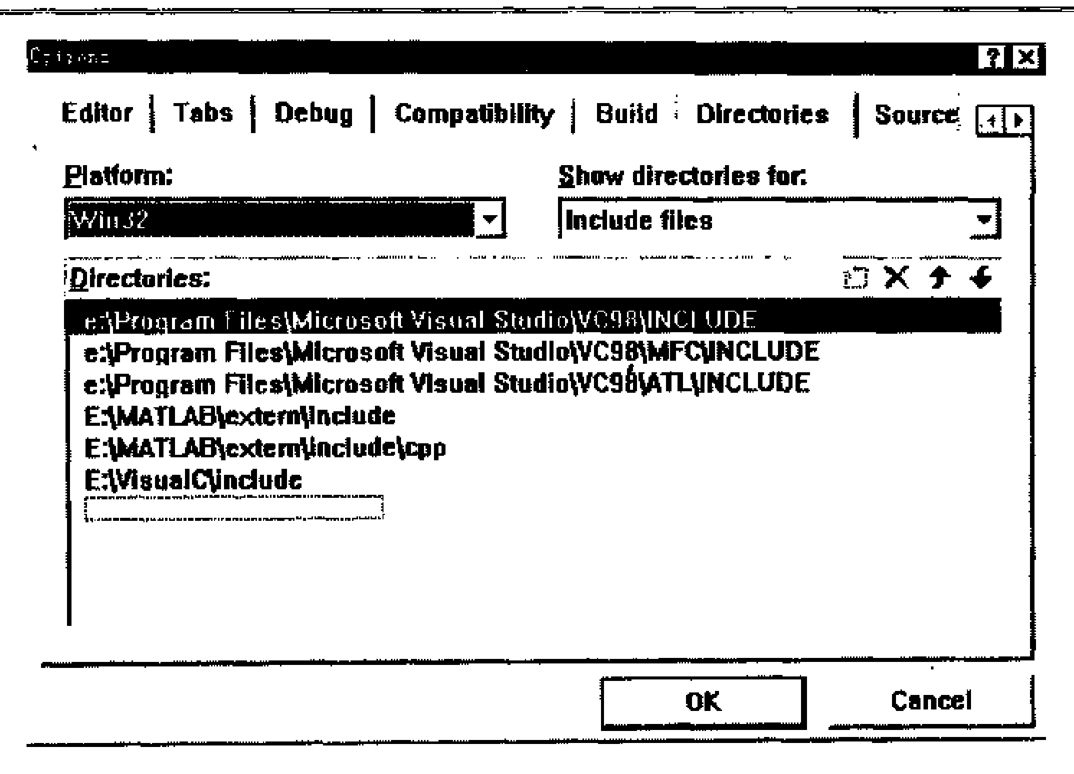


图 8.3　Options Directories 属性页

\Pogram files\Microsoft Visual Studio\VC98，并且进入该目录下的子目录\bin，按下面的格式运行该目录下的命令 lib：

```
lib /def: %MATLAB%\extern\include\libmmfile.def
                    /machine: ix86 /OUT: libmmfile.lib /NOLOGO
lib /def: %MATLAB%\extern\include\libmcc.def /machine: ix86
                    /OUT: libmcc..lib /NOLOGO
lib /def: %MATLAB%\extern\include\libmatlb.def /machine: ix86
                    /OUT: libmatlb.lib /NOLOGO
lib /def: %MATLAB%\extern\include\libmx.def /machine: ix86
                    /OUT: ibmx.lib /NOLOGO
lib /def: %MATLAB%\extern\include\libmat.def /machine: ix86
                    /OUT: libmat .lib /NOLOGO
```

执行完这五条命令后用户可以得到五个静态链接库文件，分别为 libmmfile.lib，libmcc..lib，libmatlb.lib，ibmx.lib 和 libmat.lib。命令中%MATLAB%代表本机上安装 MATLAB 的根目录，在执行这些命令的过程中，必须加以替换，如 D：\MATLAB；

这里可以明确一点，一旦五个静态链接库文件生成后，就可以反复使用，而无须对每一个项目进行重新建立；

第五步，点取下拉式菜单 Project 中的 Settings 菜单项，将弹出一个如图 8.4 所示的对话框，选取其中的 Link 属性页，在 Category 下拉框中选取"Input"，然后在 Object/Library modules 编辑框中按顺序输入在第四步中生成的五个库文件的文件名以及库文件 libmatpm.lib；

第六步，同样在 Settings 菜单项弹出的对话框中，选取 C/C++属性页，在 Category 下拉框中选取"Code Generation"，然后在 Use run-time library 下拉式选择框中选择

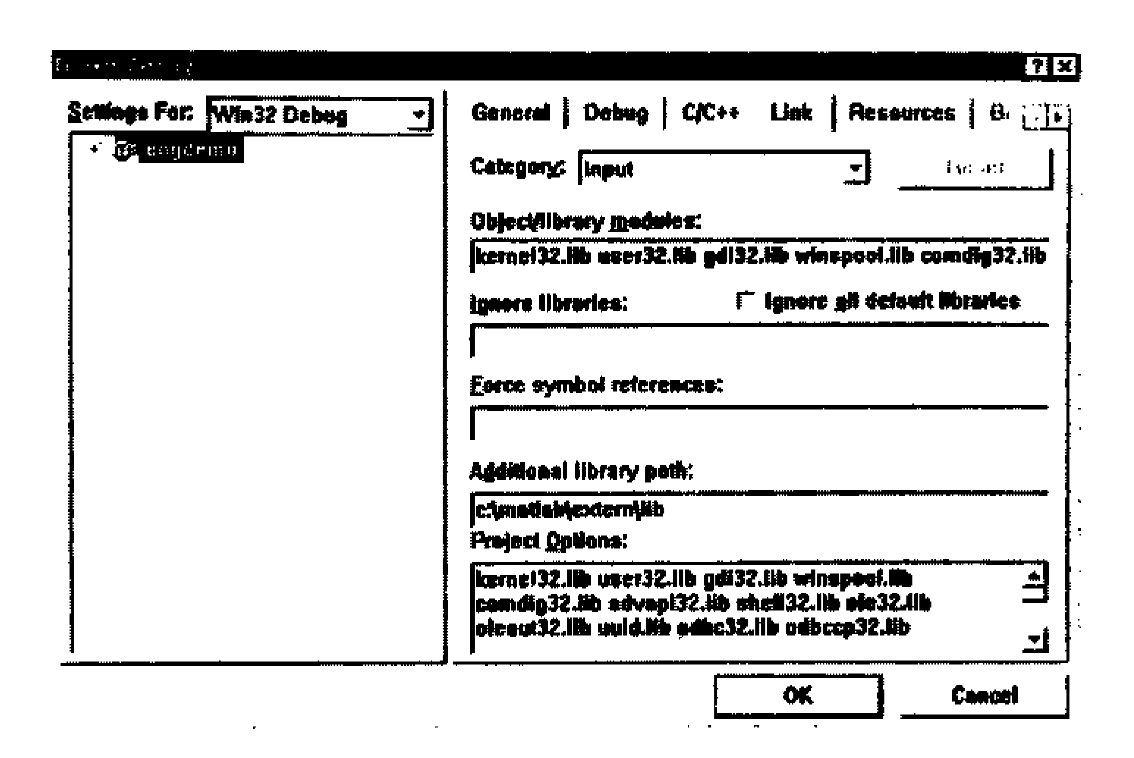


图 8.4 Project Setting Link 对话框

Multithreaded DLL 或 Debug Multithreaded DLL；这一步对于使用 MFC AppWizard 的应用程序来说是不必要的。

当完成以上六步工作之后，就可以在集成环境中对基于 MATLAB C++数学函数库的源程序进行编译和链接，生成可执行的应用程序了。这里必须注意的一点是在发布基于 VC++ 6.0 的 MATLAB C++数学函数库应用程序时必须同时附带 VC++ 6.0 提供的两个动态链接库即 msvcrt.dll 和 msvcirt.dll。

在本章中我们简要地讲述了 MATLAB C++数学函数库的基本概念和库函数的使用方法，已经基本可以满足最简单的需求，但是仍然很不深入，相当多的内容没有讲述，例如内存管理、异常处理等，有兴趣的读者可以参阅相关的书籍和联机帮助。

参 考 文 献

高俊斌. MATLAB 5.0 语言与程序设计. 1998. 武汉：华中理工大学出版社

薛定宇. 控制系统计算机辅助设计——MATLAB 语言及应用. 1996. 北京：清华大学出版社

MATLAT Application Program Interface Guide. MathWorks Inc. 1998

MATLAB Application Program Interface Reference. MathWorks Inc. 1998

MATLAB C++Math Library User's Guide. MathWorks Inc. 1999

MATLAB C++Math Library Reference. MathWorks Inc. 1999

Jerry Anderson. ACTIVEX PROGRAMMING WITH VISUAL C++. Que Corporation. 1997